TENSOR (MULTIDIMENSIONAL ARRAY) DECOMPOSITION, REGRESSION AND SOFTWARE FOR STATISTICS AND MACHINE LEARNING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

James Y. Li

May 2014

TENSOR (MULTIDIMENSIONAL ARRAY) DECOMPOSITION, REGRESSION AND SOFTWARE FOR STATISTICS AND MACHINE LEARNING

James Y. Li, Ph.D.

Cornell University 2014

This thesis illustrates connections between statistical models for tensors, introduces a novel linear model for tensors with 3 modes, and implements tensor software in the form of an `R` package. Tensors, or multidimensional arrays, are a natural generalization of the vectors and matrices that are ubiquitous in statistical modeling. However, while matrix algebra has been well-studied and plays a crucial role in the interaction between data and the parameters of any given model, algebra of higher-order arrays has been relatively overlooked in data analysis and statistical theory. The emergence of multilinear datasets - where observations are vector-variate, matrix-variate, or even tensor-variate - only serve to emphasize the relative lack of statistical understanding around tensor data structures.

In the first half of the thesis, we highlight classic tensor algebraic results and models used in image analysis, chemometrics, and psychometrics, as well as connect them to recent statistical models. The second half of the thesis features a linear model that is based off a recently introduced tensor multiplication. For this model, we prove some of the classic properties that we would expect from a 3-tensor generalization of the matrix ordinary least squares. We also apply our model to a functional dataset to demonstrate one possible usage. We conclude this thesis with an exposition of the software developed to facilitate tensor modeling and manipulation in `R`. This software implements many of the classic tensor decomposition models as well as our own linear model.

BIOGRAPHICAL SKETCH

James Li was raised by a mother who gave selflessly and a father who worked tirelessly. During his formative years, he witnessed his parents' daily struggles to establish a livelihood in the United States as working-class immigrants. Their sweat instilled deep within him a respect for hard work and understanding of the phrase "the world owes you nothing". Their joy and solidarity taught him the importance of family and being responsible for one's own actions and their consequences.

His college years at University of California, Berkeley saw a late - and at times frantic - transition from the Pre-law sophomore to the Mathematics, Statistics, and Economics triple major. The life-long friends he met there inspired a pursuit for deeper understanding of Statistics as well as a passion for all things technological.

In 2009, despite his relative inexperience and shortcomings, Cornell's Department of Statistical Science decided to give him a chance. The next five years opened his mind to the world of statistical research, brought him into the presence of well-respected scholars, and found him a life partner who shared the same ambitions.

Regardless of what the future may hold, James will always be grateful for those who have had an impact in his life. He hopes to one day be able to repay his debts to these individuals, though he knows that is probably an unattainable goal.

Dedicated to Mom, Dad, April, and Fan.

ACKNOWLEDGEMENTS

This dissertation would have been impossible without the guidance and encouragement from my advisor, Professor Martin T. Wells. You helped me narrow my focus, allowed me to sample your vast knowledge of statistics and mathematics, and offered me tremendous flexibility and support in shaping the final form of my dissertation. For all of that and more, you will always have my deepest gratitude.

The next two people I would like to thank are my Committee Members, Professor Jacob Bien and Professor James Booth. I thank you for your time and effort in thoroughly editing, critiquing, and sharpening of my research work, particularly towards the end.

My heartfelt gratitude goes out to my mentor at Yahoo! Inc., Dr. Eric T. Bax. I would not have had the same opportunities had you not first given me a chance. You have helped me again and again, and I simply do not know what I can say to do justice to the kindness you have shown me.

I owe many thanks to Professor Giles Hooker for writing me recommendation letters and for your unwavering support of the graduate students in DSS. I thank Professor David Matteson, Professor Dawn Woodard, and Professor Shane Henderson for allowing me the opportunity to work alongside and learning from you at the beginning of my graduate career. Professor Thomas J. DiCiccio has been extremely kind and helpful in offering me life advice and having me has a Teaching Assistant for the majority of the past five years. My fellow graduate students, both past and present, I am also very thankful for your help, friendship, and support.

Last but not least, I thank my family and closest friends - you know who you are - for being there when I needed you the most.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

There are two typical paths when one conducts data analysis. The first path is concerned with *making reasonable inference* from the data, drawing conclusions about the model that generated the data, and assessing said conclusions using probability distributions and/or goodness-of-fit tests. The community that favor this path generally call themselves Statisticians. The second path is more concerned with using the data to build algorithmic approaches that have the best possible *predictive accuracy*, preferably out-of-sample accuracy. The community that generally favor this second path call the process Machine Learning.

These two fields are not exclusive: Statisticians also want to make accurate predictions, and machine learners (sometimes) care about inferring underlying structure of the data generative process as well. However, the two camps can be readily distinguished by what is their primary concern. This is a statistics thesis, but tensor methodology is heavily used and influenced by image analysis - traditionally a subset of machine learning - where predictive accuracy in certain tasks (such as facial recognition) is the holy grail. As such, this thesis aims to contribute something to both communities.

Whether one is looking to do statistical inference or machine learning, one starts with the data. The shape, size, and source of the data often dictate the appropriate statistical methodology (or machine learning algorithm). Advances in medical imaging technology as well as telecommunication data-collection have ushered in massive datasets that

make multidimensional data more commonplace. The multidimensional structure of these datasets give impetus for new techniques that preserve the dimensionality of the data while still tying into the familiar framework of statistical inference and learning.

This thesis puts together various models used in image analysis under a tensor framework for statistical analysis, introduces a novel regression approach that utilize three-dimensional datasets, and develops a R package designed to facilitate the usage of tensor models amongst R users. We also address the issue of shrinkage estimation in light of our tensor regression model, and demonstrate its applications to functional data analysis.

1.1 Multilinear Data Analysis

Tensor (or multilinear) data analysis first received attention in the 1960s in psychometrics literature [53, 13, 23]. It was picked up in chemometrics beginning in the 1970s [5] and since then, tensor methods been heavily developed in that field. See Bro [10] for a recent exposition into tensor usage in chemometrics. Many papers already exist for cataloging and surveying the use of tensor techniques, with recent examples such as [28, 30, 39, 55, 20]. In particular, Kolda and Bader [30] gave a very comprehensive list of references of tensor use, noting how tensor analysis has appeared in fields such as signal processing, applied mathematics, computer vision, and data mining.

Tensor methodology has recently started to appear in statistical modeling literature as well, often under the guise of "multidimensional arrays" or "higher-order arrays". Currie

et. al. (2006) developed the "General Linear Array Model (GLAM)" for use in multi-dimensional smoothing [15]. Hoff (2011) [24] combined the tensor framework with a bayesian estimation scheme to estimate relational data. Zhou et al. (2012) [65] developed a regression model using tensor inputs and univariate responses, applying it to neuroimaging data. Zhou and Li then showed in [64] (2014) how to incorporate spectral regularization into these tensor models. Another notable development is the Population Value Decomposition model [14], developed by Crainiceanu et al. (2011). While not directly using a tensor setup, we believe (and show) that PVD reflects a variant on the general theme of a class of tensor models.

The tensor framework seems to be the correct way of generalizing the familiar notions of vectors and matrices. While flattening of the the data (treating one or more of the levels as simply more observations or more variables) and then applying traditional matrix-based methods have been proposed and often used [58, 61], methods that do not reduce the structural integrity of the data often outperform in both model parsimony and predictive performance [55, 52, 34, 33]. In fact, many of the techniques that have been developed in lieu of a formal tensor setup are later shown to be special cases of models based on the tensor structure [52], which further strengthens the claim that tensors are the natural extension to accommodate the multilinearity of today's Big Data.

These results, coupled with the rise in tensor usage in machine learning literature - both data mining [1, 60, 2, 45, 3] and computation [37, 19, 25] - warrant a much more unified framework for tensor methodology in the statistical community.

Finally, a series of papers by Kilmer et al. [26, 42, 27, 22, 62] put forth a novel way to

view and analyze tensors. We believe that this formulation of tensor multiplication based on the circulant convolution provides a new linear framework around tensors that makes it especially suitable to develop the tensor counterparts to our usual tools of projection, regression, and asymptotics. Using this novel tensor multiplication, we develop a tensor regression model that has many of the desirable properties of the Ordinary Least Squares in the case of matrix input.

1.2 Thesis Overview

The remainder of this thesis is organized as follows:

Chapter 2 will provide the relevant linear and multilinear algebra that will be used. We start by introducing the notation for the remainder of the thesis in Section 2.1, as well as the relevant matrix algebra in Section 2.2. We then provide an overview of various matrix unfolding of tensors in Section 2.3 and two major tensor multiplications in Section 2.4. We conclude the chapter with a discussion of how the two tensor multiplication differs in Section 2.5.

Chapter 3 then presents many of the most widely-used tensor decompositions and conduct a structural comparison between them and some recent statistical models that have been introduced outside the tensor context. We cover the CP decomposition in Section 3.1, the more general Tucker decomposition in Section 3.2, Generalized Low Rank Approximation (GLRAM), Multilinear Principal Component Analysis (MPCA), and 2d PCA

in Section 3.3, the Population Value Decomposition (PVD) in Section 3.4, and the Tensor Singular Value Decomposition (T-SVD) in Section 3.5.

Chapter 4 is a brief survey of multilinear tensor regression models. We cover the Tensor Regression model in Section 4.1, the Generalized Linear Array Model in Section 4.2, and end by introducing the Multilinear Normal Distribution in Section 4.3.

We motivate a novel linear model for tensors with 3 modes in Chapter 5. We first introduce the model setup in Section 5.1, then provide some algebraic results for tensors with 3 modes in Section 5.2. We then derive the least squares estimator for our model in Section 5.3 and provide an efficient FFT-based estimation algorithm in Section 5.4. We end the chapter by applying our model to a functional dataset in Section 5.5 and discussing next steps in Section 5.6.

In Chapter 6, we give a detailed look at the software we designed for tensor modeling and manipulation. In Section 6.1 we first provide an overview of available tensor software across multiple computing platforms. We then introduce `rTensor`, our own `R` [47] package in Section 6.2. We discuss our choice of the S4 Class for `rTensor` in Section 6.3. In Section 6.4, we show how to create a tensor and convert from other objects. In Section 6.5, we show how to unfold a Tensor object as well as perform tensor multiplication. In Section 6.6, we demonstrate the various tensor decompositions available in our software, and finally in Section 6.7, we demonstrate the operations specific to tensors of 3 modes.

We conclude the thesis in Chapter 7 with a summary of contributions in Section 7.1 and suggestions for future research in Section 7.2.

CHAPTER 2

TENSOR ALGEBRA

2.1 Definitions and Notation

Tensors are also known as **multidimensional arrays** or **higher-order arrays**. The modes of a tensor correspond to the dimensions of a multidimensional array. Decompositions of higher-order tensors are often called **multiway analysis** or **multilinear models**. We now give a mathematical definition.

Defintion 1. *A **tensor with** K **modes over a field** $\mathbb{F}$ **(denoted** K**-tensor**$_{\mathbb{F}}$**)** is an arranged array of numbers, where each number is a scalar from $\mathbb{F}$. The **modes of a** K**-tensor**$_{\mathbb{F}}$ are the extents or dimensions of the tensor.*

In this thesis we are primarily concerned with $\mathbb{F} = \mathbb{R}$, although there are a few instances where $\mathbb{F} = \mathbb{C}$. In those instances the distinction will be clear, so we will assume that $\mathbb{F} = \mathbb{R}$ and drop the subscript for notational simplicity from now on.

Let K denote the number of modes for a tensor, and let $n_1 \times n_2 \times \ldots \times n_K$ denote the modes or the extents associated with a K-tensor. Let n_k specify the extent of the tensor along the k^{th} mode, and let i_k denote the k^{th} index such that $1 \leq i_k \leq n_k$, $1 \leq k \leq K$.

We denote a tensor with $K \geq 3$ modes using the `\mathcal` calligraphy font, e.g. $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ is a K-tensor. We denote matrices (2-tensor) using non-bolded capital

letters, e.g. $X \in \mathbb{R}^{n_1 \times n_2}$. We denote vectors (1-tensors) using bolded lower case letters, e.g. $\mathbf{x} \in \mathbb{R}^{n_1}$ and scalars (0-tensors) using non-bolded lower case letters, e.g. $x \in \mathbb{R}$.

Since we work mostly with 3-tensors in this thesis, it is also beneficial to define the following. Let a **slice** of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ be a matrix obtained by fixing one index of $\mathcal{X}$ and leaving the remaining two free. Let a **tube** of $\mathcal{X}$ be a vector obtained by fixing two indices and leaving one free. Denote free indices using : and subsetting operations of a tensor using brackets following the tensor.

For instance, $\mathcal{X}[:, :, 1]$ is the first slice of $\mathcal{X}$ along the third mode, and $\mathcal{X}[:, 5, :]$ is the fifth slice of $\mathcal{X}$ along the second mode. Also $\mathcal{X}[:, 3, 4]$ is a vector obtained by fixing $n_2 = 3$ and $n_3 = 4$.

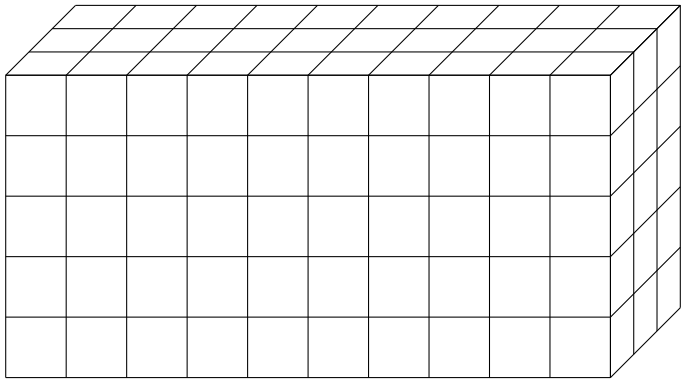


Figure 2.1: This cuboid helps to visualize a 3-tensor $\mathcal{X} \in \mathbb{R}^{5 \times 10 \times 3}$, with each small cube containing a scalar in $\mathbb{R}$.

The **Frobenius norm** of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ extends the matrix case in the usual manner:

$$\|\mathcal{X}\|_F^2 := \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_K=1}^{n_K} x_{i_1,\ldots,i_K}^2.$$

The **inner product** between two tensors of the same modes also extends the matrix case in the usual manner. Let $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, then

$$\langle \mathcal{X}, \mathcal{Y} \rangle := \sqrt{\sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \ldots \sum_{i_K=1}^{n_K} x_{i_1,\ldots,i_K} y_{i_1,\ldots,i_K}}.$$

Addition and subtraction are defined element-wise for tensors of the same modes. That is, given $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$:

$$(\mathcal{X} \pm \mathcal{Y})_{i_1,\ldots,i_K} := \mathcal{X}_{i_1,\ldots,i_K} \pm \mathcal{Y}_{i_1,\ldots,i_K}, \ , 1 \leq i_k \leq n_k, 1 \leq k \leq K.$$

Note that although these definitions have been provided for real-valued tensors, complex-valued tensors also have the same properties.

2.2 Relevant Matrix Algebra

In this section, we provide formal definitions for the matrix operations and structures that play crucial roles in understanding tensor operations from a matrix perspective.

Defintion 2. *If $A \in \mathbb{R}^{n_1 \times n_2}$ and $B \in \mathbb{R}^{n_3 \times n_4}$, then the **Kronecker product** of A and B is:*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \ldots & a_{1n_1}B \\ a_{21}B & a_{22}B & \ldots & a_{2n_1}B \\ \vdots & \vdots & \vdots & \vdots \\ a_{m_1 1}B & a_{m_1 2}B & \ldots & a_{m_1 n_1}B \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times n_2 n_4}.$$

Now if A, B had the same number of columns (i.e. $n = n_2 = n_4$), then the ***Khatri-Rao product*** *of A and B is defined to be the column-wise Kronecker product:*

$$A \odot B = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \dots \mathbf{a}_n \otimes \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times n}.$$

Finally if A, B had the exact same dimensions (i.e. $m = n_1 = n_3, n = n_2 = n_4$), then the ***Hamadard product*** *of A and B is defined to be the element-wise product:*

$$A * B = \begin{bmatrix} a_{11}b_{11} & \dots & a_{1n}b_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1}b_{m1} & \dots & a_{mn}b_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

We also define another special matrix structure known as the circulant matrix that is relevant for both tensor decompositions and regression. A circulant matrix $A \in \mathbb{R}^{n \times n}$ is a special type of Topelitz matrix where each column of A can be obtained by shifting the previous column down one value.

Defintion 3. *A* ***circulant matrix*** *$A \in \mathbb{R}^{n \times n}$ is a square matrix fully specified by a vector of length n, $\mathbf{v} = \begin{pmatrix} v_1 & v_2 & v_3 \dots & v_n \end{pmatrix}^T$:*

$$A = \text{circ}(\mathbf{v}) = \begin{bmatrix} v_1 & v_n & v_{n-1} & \dots & v_2 \\ v_2 & v_1 & v_n & \dots & v_3 \\ v_3 & v_2 & v_1 & \dots & v_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_n & v_{n-1} & v_{n-2} & \dots & v_1 \end{bmatrix}$$

The circulant matrix has the special property that it can be diagonalized by the Discrete

Fourier Transform (DFT), a widely used transform. We use the definition from a classical text [9].

Defintion 4. *Let* $\mathbf{v} = \begin{pmatrix} v_1 & v_2 & v_3 \dots & v_n \end{pmatrix} \in \mathbb{R}^n$*, then the* ***Discrete Fourier Transform*** *of* $\mathbf{v}$ *is the sequence* $\mathbf{f} \in \mathbb{R}^n$*, where*

$$\mathbf{f}_j = \sum_{k=0}^{n-1} v_k e^{-i2\pi jk/n}.$$

The ***Fast Fourier Transform*** *(FFT), denoted fft(*$\mathbf{v}$*), computes the DFT quickly and efficiently. The FFT of* $\mathbf{v}$ *can also be seen as a left multiplication by the Vandermonde matrix*

$$F_n = \begin{bmatrix} \omega_n^{0\cdot 0} & \omega_n^{0\cdot 1} & \dots & \omega_n^{0\cdot(n-1)} \\ \omega_n^{1\cdot 0} & \omega_n^{1\cdot 1} & \dots & \omega_n^{1\cdot(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_n^{(n-1)\cdot 0} & \omega_n^{(n-1)\cdot 1} & \dots & \omega_n^{(n-1)\cdot(n-1)} \end{bmatrix},$$

where $\omega_n = e^{-i2\pi/n}$*. The inverse FFT, denoted iff(*$\mathbf{f}$*), can be seen as a left multiplication by the complex conjugate of* F_n*, which we denote* F_n^**. Hence we have* $\mathbf{f}_j = (F_n \cdot \mathbf{v})_j$ *and* $\mathbf{v}_j = (F_n^* \cdot \mathbf{f})_j$*.*

A circulant matrix A specified by $\mathbf{v} \in \mathbb{R}^n$ can be diagonalized by F_n as follows:

$$F_n^* \cdot A \cdot F_n = \frac{1}{n} \begin{bmatrix} \mathbf{f}_1 & & & \\ & \mathbf{f}_2 & & \\ & & \ddots & \\ & & & \mathbf{f}_n \end{bmatrix}$$

Now with a series of matrices $A_1, A_2, \dots, A_n$, we can similarly construct a block circulant matrix by shifting down 1 matrix at a time for each block.

Defintion 5. *A **block circulant matrix*** $\mathbf{A} \in \mathbb{R}^{n_1 n_3 \times n_2 n_3}$ *is fully specified by a series of* n_3 *matrices* $A_1, A_2, \ldots, A_{n_3}$*, each* $A_j \in \mathbb{R}^{n_1 \times n_2}$*:*

$$\mathbf{A} = \begin{bmatrix} A_1 & A_{n_3} & A_{n_3-1} & \ldots & A_2 \\ A_2 & A_1 & A_{n_3} & \ldots & A_3 \\ A_3 & A_2 & A_1 & \ldots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n_3} & A_{n_3-1} & A_{n_3-2} & \ldots & A_1 \end{bmatrix}.$$

Similar to the circulant matrix, the block circulant matrix can be block diagonalized by F_n [27]:

$$(F_{n_3}^* \otimes I_{n_1}) \cdot \mathbf{A} \cdot (F_{n_3} \otimes I_{n_2}) = \frac{1}{n} \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_{n_3} \end{bmatrix},$$

where each $D_j \in \mathbb{C}^{n_1 \times n_2}$ has a special structure that is discussed in more depth in Section 2.4.2.

2.3 Tensor Unfolding

For $K \geq 3$, it is often useful to be able to represent a K-tensor as a matrix or as a vector, especially as a first step in defining a tensor multiplication. This representation is often called unfolding or flattening. As we will see in Section 2.4, tensor unfolding allows us to define tensor products using familiar matrix operations. In this section, we first elaborate

on the tensor unfolding operations that have been used predominantly in tensor analysis, and then describe a general way of thinking about tensor unfolding that connects these seemingly disparate operations.

First consider the vector representation of a K-tensor, which is simply a stacking of the K-tensor element-wise into a $n_1 n_2 \dots n_K$ vector. A useful convention is to allow the last index to vary the fastest and the first index to vary the slowest. This prompts the following natural definition.

Defintion 6. *For a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_K}$, the **vectorization** of $\mathcal{X}$, denoted* $\text{vec}(\mathcal{X})$*, is the operation that creates a vector of length $n_1 n_2 \dots n_K$ from the elements of $\mathcal{X}$, ordered according to the convention that allows i_a to vary faster than i_b for $K \geq a > b \geq 1$.*

For instance, for a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$,

$$\text{vec}(\mathcal{X}) = \begin{pmatrix} x_{111} \\ x_{112} \\ \vdots \\ x_{121} \\ x_{122} \\ \vdots \\ x_{n_1 n_2 n_3} \end{pmatrix} \in \mathbb{R}^{n_1 n_2 n_3}.$$

Now consider how a general K-tensor can be represented in matrix form. One definition that has prevailed in earlier tensor literature is the k-mode matricization/unfolding [28].

Defintion 7. *For $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, denote $\mathcal{X}_{(k)} \in \mathbb{R}^{n_k \times \prod_{j \neq k} n_j}$ to be the k-**mode unfolding**. $\mathcal{X}_{(k)}$ is a mapping from the $(i_1, i_2, \ldots, i_K)^{th}$ element to the $(i_k, j)^{th}$ element of the resulting matrix, where*

$$j = 1 + \sum_{p \neq k}^{K} (i_p - 1) J_p, \text{ with } J_p = \prod_{q \neq k}^{p-1} n_q,$$

ordered according to the convention that allows i_a to vary faster than i_b for $K \geq a > b \geq 1$.

The convention in the permutation of the indices $\{n_1, \ldots, n_{k-1}, n_{k+1}, \ldots, n_K\}$ is consistent to the convention from the vec$(\cdot)$ operation. For a 3-tensor, there are three k-mode unfoldings, denoted $\mathcal{X}_{(1)}$, $\mathcal{X}_{(2)}$, and $\mathcal{X}_{(3)}$, all of which are demonstrated in Figure 2.2.

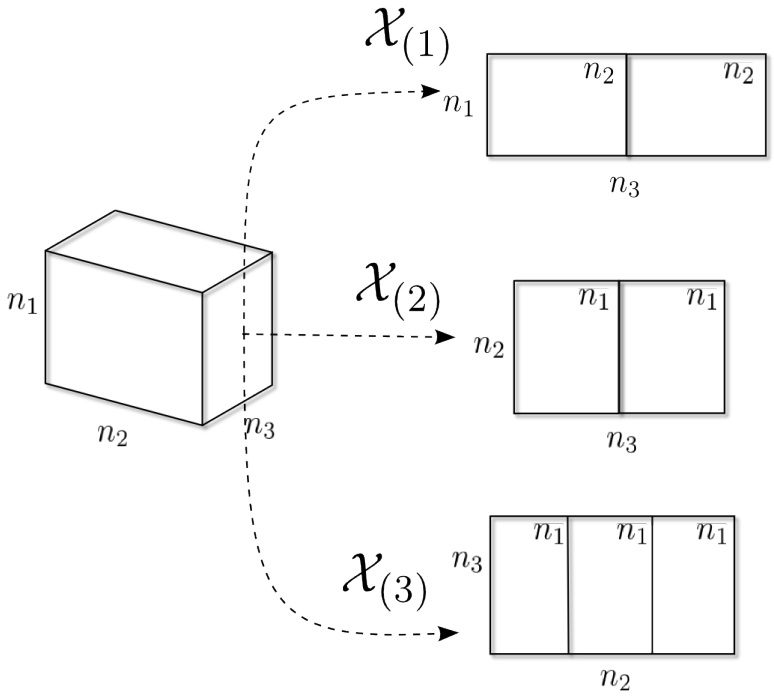


Figure 2.2: 1-mode, 2-mode, and 3-mode unfoldings for a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

Kilmer et. al. proposed another tensor unfolding known as the matvec(·) operation [27]. Similar to how vec(·) of a matrix stacks its columns to form a vector, matvec(·) of a 3-tensor stacks the slices (2-tensors) along the third mode and stacks them to form a matrix.

Defintion 8. *For* $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$*, let* $X_j := \mathcal{X}[:, :, j]$*,* $j = 1, \ldots, n_3$*, then the* ***matrix vectorization*** *of* $\mathcal{X}$*, denoted* $\text{matvec}(\mathcal{X})$*, is*

$$\text{matvec}(\mathcal{X}) = \begin{bmatrix} X_1 \\ \vdots \\ X_{n_3} \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times n_2}.$$

The matvec is more explicitly illustrated in Figure 2.3.

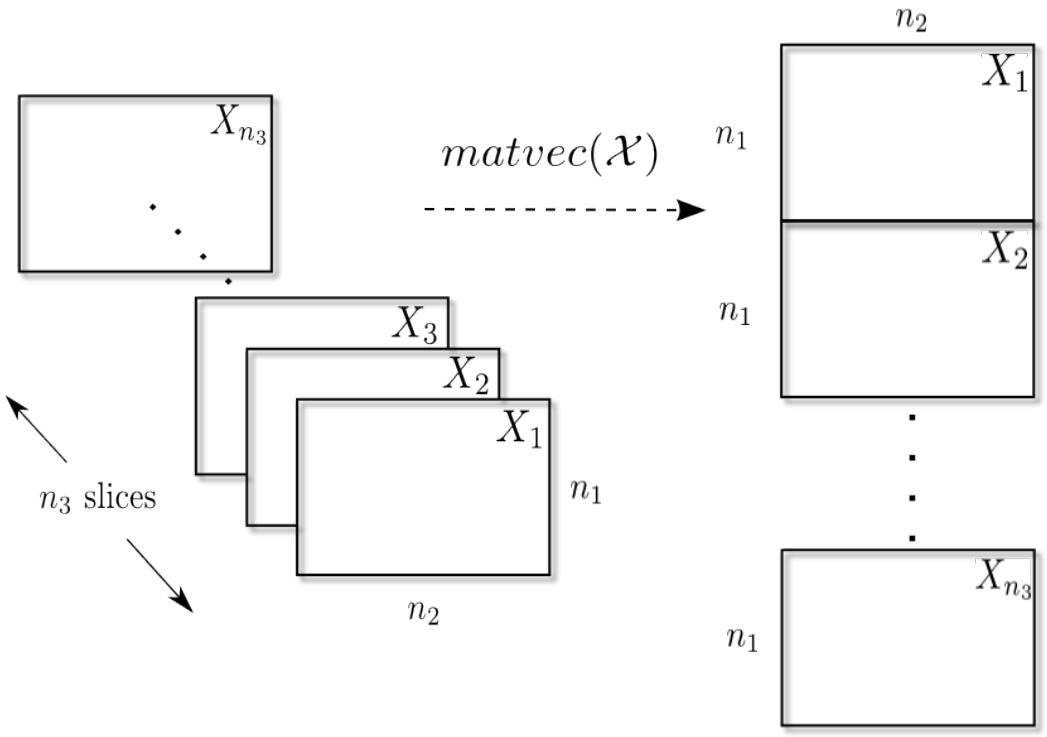


Figure 2.3: matvec($\mathcal{X}$) for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ which results in a $n_1 n_3 \times n_2$ matrix,

While matvec$(\cdot)$ is also defined for general K-tensors[42], it does not result in a matrix, so we do not elaborate on that in this paper.

We now define a more general way of considering tensor unfolding which provides additional insight into the differences between the matvec$(\cdot)$ and k-mode unfolding. We first define a K-tensor generalization of a tube.

Defintion 9. *For any $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, let the **k-mode vectors** be the vectors indexed by all indices except i_k. There are $n_1 n_2 \ldots n_{k-1} n_{k+1} \ldots n_K$ vectors of length n_k along the k-th*

mode. Denote these as

$$\mathcal{X}[i_1, \ldots, i_{k-1}, :, i_{k+1}, \ldots, i_K] \in \mathbb{R}^{n_k}, 1 \leq i_j \leq n_k, 1 \leq k \leq K.$$

We can now define the row/column space unfolding of a tensor.

Defintion 10. *The **row space unfolding of** $\mathcal{X}$ **in the mode** k, denoted* $\text{unfold}_{RS}(\cdot, k)$*, is the stacking of all k-mode vectors of $\mathcal{X}$ as columns in the resulting matrix. Similarly, the **column space unfolding of** $\mathcal{X}$ **in the mode** k, denoted* $\text{unfold}_{CS}(\cdot, k)$*, is the stacking of all k-mode vectors as rows in the resulting matrix. For any $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$,*

$$\text{unfold}_{RS}(\mathcal{X}, k) \in \mathbb{R}^{n_k \times (\prod_{j \neq k} n_j)}, \text{ and}$$

$$\text{unfold}_{CS}(\mathcal{X}, k) \in \mathbb{R}^{(\prod_{j \neq k} n_j) \times n_k}.$$

Furthermore, $(\text{unfold}_{RS}(\mathcal{X}, k))^T = \text{unfold}_{CS}(\mathcal{X}, k)$.

As an example, consider $\mathcal{X} \in \mathbb{R}^{3 \times 4 \times 5}$; the row space unfoldings are:

$$\text{unfold}_{RS}(\mathcal{X}, 1) = \underbrace{\begin{bmatrix} \mathcal{X}[:,1,1] & \mathcal{X}[:,1,2] & \ldots & \mathcal{X}[:,1,5] & \mathcal{X}[:,2,1] & \ldots & \mathcal{X}[:,4,5] \end{bmatrix}}_{\text{20 columns vectors of length 3}}$$

$$\text{unfold}_{RS}(\mathcal{X}, 2) = \underbrace{\begin{bmatrix} \mathcal{X}[1,:,1] & \mathcal{X}[1,:,2] & \ldots & \mathcal{X}[1,:,5] & \mathcal{X}[2,:,1] & \ldots & \mathcal{X}[3,:,5] \end{bmatrix}}_{\text{15 columns vectors of length 4}}$$

$$\text{unfold}_{RS}(\mathcal{X}, 3) = \underbrace{\begin{bmatrix} \mathcal{X}[1,1,:] & \mathcal{X}[1,2,:] & \ldots & \mathcal{X}[1,4,:] & \mathcal{X}[2,1,:] & \ldots & \mathcal{X}[3,4,:] \end{bmatrix}}_{\text{12 columns vectors of length 5}},$$

and the columns space unfoldings are:

$$\text{unfold}_{CS}(\mathcal{X}, 1) = \underbrace{\begin{bmatrix} \mathcal{X}[:,1,1]^T \\ \mathcal{X}[:,1,2]^T \\ \cdots \\ \mathcal{X}[:,1,5]^T \\ \mathcal{X}[:,2,1]^T \\ \cdots \\ \mathcal{X}[:,4,5]^T \end{bmatrix}}_{\text{20 row vectors of length 3}}$$

$$\text{unfold}_{CS}(\mathcal{X}, 2) = \underbrace{\begin{bmatrix} \mathcal{X}[1,:,1]^T \\ \mathcal{X}[1,:,2]^T \\ \cdots \\ \mathcal{X}[1,:,5]^T \\ \mathcal{X}[2,:,1]^T \\ \cdots \\ \mathcal{X}[3,:,5]^T \end{bmatrix}}_{\text{15 row vectors of length 4}}$$

$$\text{unfold}_{CS}(\mathcal{X}, 3) = \underbrace{\begin{bmatrix} \mathcal{X}[1,1,:]^T \\ \mathcal{X}[1,2,:]^T \\ \dots \\ \mathcal{X}[1,4,:]^T \\ \mathcal{X}[2,1,:]^T \\ \dots \\ \mathcal{X}[3,4,:]^T \end{bmatrix}}_{\text{12 row vectors of length 5}}.$$

These two definitions allow us to better juxtapose the k-mode unfolding of a 3-tensor $\mathcal{X}$ and the matvec$(\cdot)$, since the former is exactly the same as the row space unfolding in the k^{th} mode while the latter is exactly the same as column space unfolding in the second mode. Hence, the main difference between these two unfolding operations can be attributed to whether we stack the k-mode vectors as rows or columns.

The folding operations, which invert these unfold operations, are defined through the unfolding themselves. It is important to note that the foldings operate on any arbitrary matrix, so it becomes necessary to specify the exact modes of the resulting tensor.

Defintion 11. *Let the folding operation of matrices into tensors as the inverse operations to the corresponding tensor unfolding, and let* $mn = \prod_{k=1}^{K} n_k$. *The* ***row space folding of a matrix in the mode*** k *for* $X \in \mathbb{R}^{m \times n}$ *is*

$$\text{fold}_{RS}(X, k, n_1 \times n_2 \times \ldots \times n_K) = \mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K},$$

where $X = \text{unfold}_{RS}(\mathcal{X}, k)$. *Similarly, define* ***columnn space folding*** *of* $X \in \mathbb{R}^{m \times n}$ *to be*

$$\text{fold}_{CS}(X, k, n_1 \times n_2 \times \ldots \times n_K) = \mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K},$$

where $X = \text{unfold}_{CS}(\mathcal{X}, k)$.

Another way to explicitly represent these tensor unfoldings is with the use of basis vectors and Kronecker notation. First let $e_j \in \mathbb{R}^{1\times n}$ be the j^{th} column unit basis vector of length n and $e_j^T \in \mathbb{R}^{n\times 1}$ be the row unit basis vector of length n. For notational convenience, we dropped the length of the basis vectors where it is unambiguous to do so.

Now any matrix $X \in \mathbb{R}^{n_1\times n_2}$ can be written as a sum of its scalar elements:

$$X = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}(e_i \otimes e_j^T)X_{ij}.$$

Furthermore, we can now write down the $\text{vec}(X) \in \mathbb{R}^{n_1 n_2}$ as

$$\text{vec}(X) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}(e_i \otimes e_j)X_{ij}.$$

We see that the row and column basis vectors allow us to explicitly "pick off" the individual elements of the matrix and re-arrange them.

Now for $\mathcal{X} \in \mathbb{R}^{n_1\times n_2\times n_3}$, we can use these to explicitly write down the three k-mode unfoldings of the 3-tensor:

$$\begin{aligned}
\mathcal{X}_{(1)} &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}(e_i \otimes e_j^T \otimes e_k^T)\mathcal{X}_{ijk} \\
\mathcal{X}_{(2)} &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}(e_i^T \otimes e_j \otimes e_k^T)\mathcal{X}_{ijk} \\
\mathcal{X}_{(3)} &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}(e_i^T \otimes e_j^T \otimes e_k)\mathcal{X}_{ijk},
\end{aligned}$$

as well as the matvec(·) and vec(·):

$$\text{matvec}(\mathcal{X}) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}(e_i \otimes e_j^T \otimes e_k)\mathcal{X}_{ijk},$$
$$\text{vec}(\mathcal{X}) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3}(e_i \otimes e_j \otimes e_k)\mathcal{X}_{ijk}.$$

This extends to general K-tensors and the row/column space definition as well:

$$\text{unfold}_{RS}(\mathcal{X}, k) = \sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\dots\sum_{i_K=1}^{n_K}(e_{i_1}^T \otimes e_{i_2}^T \otimes \dots \otimes e_{i_{k-1}}^T \otimes e_{i_k} \otimes e_{i_{k+1}}^T \otimes \dots e_{i_K}^T)\mathcal{X}_{i_1,\dots,i_K}$$
$$\text{unfold}_{CS}(\mathcal{X}, k) = \sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\dots\sum_{i_K=1}^{n_K}(e_{i_1} \otimes e_{i_2} \otimes \dots \otimes e_{i_{k-1}} \otimes e_{i_k}^T \otimes e_{i_{k+1}} \otimes \dots e_{i_K})\mathcal{X}_{i_1,\dots,i_K}$$
$$\text{vec}(\mathcal{X}) = \sum_{i_1=1}^{N_1}\sum_{i_2=1}^{N_2}\dots\sum_{i_K=1}^{N_K}(e_{i_1} \otimes e_{i_2} \otimes \dots \otimes e_{i_k})\mathcal{X}_{i_1,\dots,i_K}$$

2.4 Tensor Multiplication

In this section, we discuss two prevailing definitions of tensor products - the k-mode multiplication and the t-product - and illustrate the crucial differences.

2.4.1 k-mode Product

We start with the definition.

Defintion 12. *The k-**mode product** specifies multiplication between a K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_K}$ and a matrix $M \in \mathbb{R}^{J \times n_k}$, where n_k is the k mode for $\mathcal{X}$ [28]. The result is*

a K-tensor in $\mathbb{R}^{n_1 \times \ldots \times n_{k-1} \times J \times n_{k+1} \times \ldots \times n_K}$. *This is defined element-wise to be:*

$$(\mathcal{X} \times_k M)_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_K} = \sum_{i_k=1}^{n_k} \mathcal{X}_{i_1,\ldots,i_K} M_{j,i_k}.$$

As the name suggests, this product definition is closely related to the k-mode unfolding. In fact:

$$\mathcal{Y} = \mathcal{X} \times_k M \quad \Leftrightarrow \quad \mathcal{Y}_{(k)} = M \cdot \mathcal{X}_{(k)},$$

where $\cdot$ denotes the usual matrix multiplication.

In other words, we can think about the k-mode product as a left matrix multiplication onto the k-mode vectors: each k-mode vector of the resulting tensor $\mathcal{Y}$ is a result of a matrix-vector multiplication between M and the corresponding k-mode vector of $\mathcal{X}$. Note that if M is a vector (i.e. $J = 1$), then each k-mode vector of $\mathcal{Y}$ is the result of an inner product between two vectors, and $\mathcal{Y}$ will have $n_k = 1$ and is essentially a $(K-1)$-tensor.

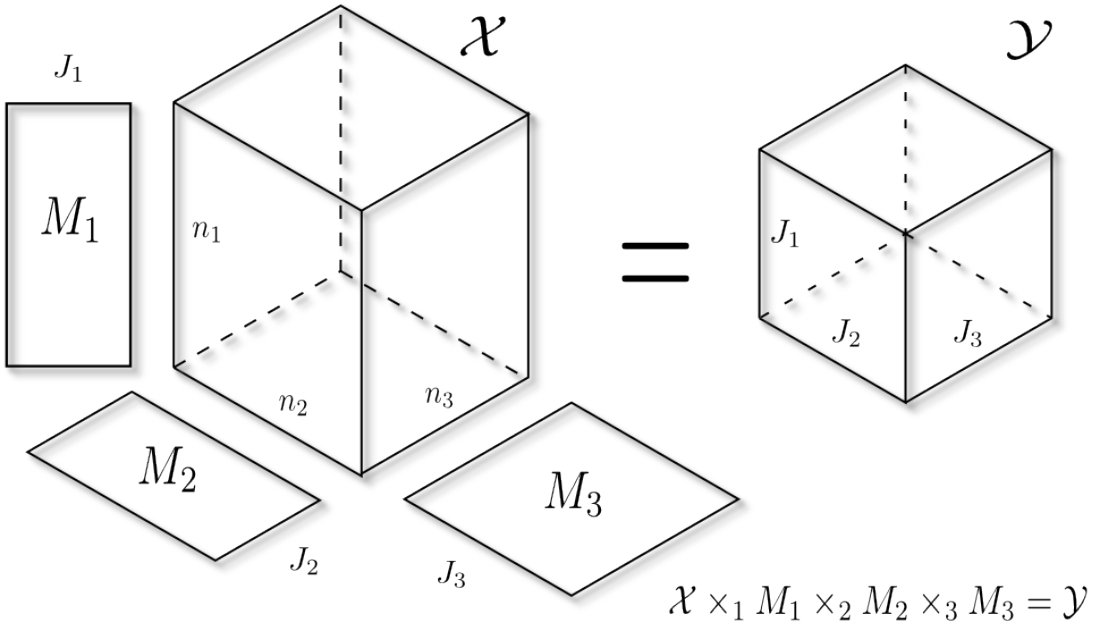


Figure 2.4: k-mode product of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and matrices $M_k \in \mathbb{R}^{J_k \times n_k}$. The result is $\mathcal{Y} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$.

Also note that if $X \in \mathbb{R}^{n_1 \times n_2}$ were a matrix, then the k-mode product between X and $M_1 \in \mathbb{R}^{J_1 \times n_1}, M_2 \in \mathbb{R}^{J_2 \times n_2}$ is equivalent to the following matrix products:

$$X \times_1 M_1 = M_1^T \cdot X \in \mathbb{R}^{J_1 \times n_2}$$

$$X \times_2 M_2 = X \cdot M_2 \in \mathbb{R}^{n_1 \times J_2}$$

The k-mode product serves as the basis for many tensor decompositions and regression models, including the Tucker decomposition and the CP decomposition. It also bears mentioning that the Kronecker product permits a matrix view of the product between a

general K-tensor and a list of matrices [28]:

$$\mathcal{Y} = \mathcal{X} \times_1 M_1 \times_2 M_2 \ldots \times_K M_K$$
$$\Leftrightarrow \mathcal{Y}_{(k)} = M_k \cdot \mathcal{X}_{(k)} \cdot (M_K \otimes \ldots \otimes M_{k+1} \otimes M_{k-1} \ldots \otimes M_1)^T$$

For more properties of the k-mode product, see [28].

2.4.2 t-Product

While the k-mode product defines multiplication between a tensor and a matrix, it does not provide a natural way to multiply two 3-tensors. To this end, the t-product has recently been proposed by [27]. We believe this latter tensor product shows great promise in its applicability to statistical modeling and regression. Prior to exploration of the t-product, however, we need to recall the block circulant matrix structure from Section 2.2.

We can couple the block circulant structure with the matvec$(\cdot)$ operation to create a block circulant matrix using the slices of a 3-tensor along mode 3. Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then:

$$\text{circ}(\text{matvec}(\mathcal{X})) = \begin{bmatrix} X_1 & X_{n_3} & X_{n_3-1} & \ldots & X_2 \\ X_2 & X_1 & X_{n_3} & \ldots & X_3 \\ X_3 & X_2 & X_1 & \ldots & X_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_{n_3} & X_{n_3-1} & X_{n_3-2} & \ldots & X_1 \end{bmatrix},$$

where $X_j = \mathcal{X}[:, :, j]$ is defined to the the j^{th} slice of $\mathcal{X}$ along mode 3. Furthermore, this

means that for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$,

$$(F_{n_3}^* \otimes I_{n_1}) \cdot \text{circ}(\text{matvec}(\mathcal{X})) \cdot (F_{n_3} \otimes I_{n_2}) = diag(D_1, \ldots, D_{n_3}),$$

where F_n is the n^{th} order discrete Fourier transform matrix and $D_j \in \mathbb{C}^{n_1 \times n_2}$. It is important to note that $D_j = \mathcal{D}[:, :, j]$, where $\mathcal{D} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is created via $n_1 \times n_2$ FFT's: $\mathcal{D}[i_1, i_2, :] = F_{n_3}(\mathcal{X}[i_1, i_2, :])$. The t-product is defined via the block circulant structure and the matvec$(\cdot)$ operator and allows for a direct multiplication of two 3-tensors.

Defintion 13. *For $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$, the **t-product** is $\mathcal{A} * \mathcal{B} \in \mathbb{R}^{n_1 \times L \times n_3}$, where the* matvec$(\mathcal{A} * \mathcal{B})$ *is a result of matrix multiplication:*

$$\begin{aligned} \text{matvec}(\mathcal{A} * \mathcal{B}) &= \text{circ}(\text{matvec}(\mathcal{A})) \cdot matvec(\mathcal{B}) \\ &= \begin{bmatrix} A_1 & A_{n_3} & A_{n_3-1} & \ldots & A_2 \\ A_2 & A_1 & A_{n_3} & \ldots & A_3 \\ A_3 & A_2 & A_1 & \ldots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n_3} & A_{n_3-1} & A_{n_3-2} & \ldots & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_{n_3} \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times L}. \end{aligned} \tag{2.1}$$

Here $A_j = \mathcal{A}[:, :, j], B_j = \mathcal{B}[:, :, j]$ are the j^{th} slices of $\mathcal{A}$ and $\mathcal{B}$ along mode 3 respectively. To get the tensor $\mathcal{A} * \mathcal{B}$, we simply have to fold matvec$(\mathcal{A} * \mathcal{B})$ *using the inverse folding for* matvec$(\cdot)$. *Since we noted that the* matvec$(\cdot)$ *operation is equivalent to* unfold$_{CS}(\cdot, 2)$, *then we have $\mathcal{A} * \mathcal{B} =$* fold$_{CS}$(matvec$(\mathcal{A} * \mathcal{B}), 2, n_1 \times L \times n_3)$.

From the definition of the t-product, we can see that each mode-3 slice of the resulting tensor $\mathcal{A} * \mathcal{B}$ is given by a sum of products of the mode-3 slices of $\mathcal{A}$ and $\mathcal{B}$. In fact, the t-product $\mathcal{A} * \mathcal{B}$ defines a linear map that takes $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$ to $\mathbb{R}^{n_1 \times L \times n_3}$ [42]. It also

allows the extension of familiar linear algebra concepts such as the transpose, orthgonality, nullspace, and range. For detailed accounts of these properties, refer to [27]. When $n_3 = 1$, we get back the usual matrix multiplication.

For general K-tensors, where $K \geq 4$, the t-product is extended in [42] to be defined recursively with the base case being the 3-tensor t-product. For instance, for 4-tensors $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}, \mathcal{B} \in \mathbb{R}^{n_3 \times L \times n_3 \times n_4}$, we have

$$\mathcal{A} * \mathcal{B} := \text{fold}_{CS}\left(\begin{bmatrix} \mathcal{A}_1 & \mathcal{A}_{n_3} & \mathcal{A}_{n_3-1} & \dots & \mathcal{A}_2 \\ \mathcal{A}_2 & \mathcal{A}_1 & \mathcal{A}_{n_3} & \dots & \mathcal{A}_3 \\ \mathcal{A}_3 & \mathcal{A}_2 & \mathcal{A}_1 & \dots & \mathcal{A}_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}_{n_3} & \mathcal{A}_{n_3-1} & \mathcal{A}_{n_3-2} & \dots & \mathcal{A}_1 \end{bmatrix} * \begin{bmatrix} \mathcal{B}_1 \\ \mathcal{B}_2 \\ \mathcal{B}_3 \\ \vdots \\ \mathcal{B}_{n_3} \end{bmatrix}\right) \in \mathbb{R}^{n_1 \times L \times n_3 \times n_4},$$

where each $\mathcal{A}_j \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\mathcal{B}_j \in \mathbb{R}^{n_2 \times L \times n_3}$ are the j^{th} sub-tensors of $\mathcal{A}$ and $\mathcal{B}$ along the 4^{th} mode respectively, and $\mathcal{A}_i * \mathcal{B}_j$ is defined as in Equation 2.1, $\forall i, j \in \{1, \dots, n_4\}$. Once again, the t-product has the characteristic that every sub-tensor of $\mathcal{B}$ is hit by every sub-tensor of $\mathcal{A}$.

Our earlier note regarding the DFT block-diagonalization plays a crucial role in increasing the computation efficiency of the t-product, as it reduces both time and storage costs. Using the DFT, one does not need to construct the full block circulant matrix from $\mathcal{A}$ in the operation $\mathcal{A} * \mathcal{B}$, as shown below.

Algorithm 1 t-Product for 3-Tensors

input : $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\tilde{\mathcal{A}}[i_1, i_2, :] = \text{fft}(\mathcal{A}[i_1, i_2, :])$
  end
  for $i_2 = 1, \ldots, L$ **do**
    $\tilde{\mathcal{B}}[i_1, i_2, :] = \text{fft}(\mathcal{B}[i_1, i_2, :])$
  end
end
for $j = 1, \ldots, n_3$ **do**
  $\tilde{C}[:, :, j] = \tilde{\mathcal{A}}[:, :, j] \cdot \tilde{\mathcal{B}}[:, :, j]$
end
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, L$ **do**
    $C[i_1, i_2, :] = \text{ifft}(C[i_1, i_2, :])$
  end
end
output : $C = \mathcal{A} * \mathcal{B} \in \mathbb{R}^{n_1 \times L \times n_3}$

The DFT block-diagonalization technique also makes tensor decompositions involving the t-product much more efficient. Note however, that the the DFT transform is not necessary in the definition of t-product; it is only a computation aid to increase both speed and storage efficiency of the operation.

2.5 Linear versus Multilinear

A natural question to ask about the two different tensor products defined above is how do they differ? Aside from the obvious structural differences between the two, the crucial

distinction is that while the t-product defines a *linear map* for a K-tensor, the k-mode product defines a *multilinear map* [27] for a list of matrices.

To be explicit, consider the case where $K = 3$. The t-product between $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$ defines a linear map $\mathbb{R}^{n_2 \times L \times n_3} \mapsto \mathbb{R}^{n_1 \times L \times n_3}$ via the operation $\mathcal{A} * \mathcal{B}$. When $n_3 = 1$, then the t-product reduces to the usual matrix product between $A \in \mathbb{R}^{n_1 \times n_2}$ and $B \in \mathbb{R}^{n_2 \times L}$, which is a linear map for $\mathbb{R}^{n_2 \times L} \mapsto \mathbb{R}^{n_1 \times L}$ via the operation $A \cdot B$.

On the other hand, a multilinear map takes multiple arguments and is linear in each of the arguments*separately*. The k-mode product between $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and a list of matrices $M_k \in \mathbb{R}^{J_k \times n_k}, 1 \leq k \leq 3$, defines a map that is linear in each $\mathbb{R}^{J_k \times n_k} \mapsto \mathbb{R}^{J_k \times \prod_{j \neq k} n_j}$ separately via a change of basis on $\mathcal{A} \times_k M_k$.

A consequence of this distinction is that the k-mode product treats all the modes of the original tensor in exactly the same way, while t-product does not; the ordering of the modes on the 3-tensor matters for the t-product. This should not be surprising as ordering also matters when it comes to matrix multiplication. As we will see in the next section, both the t-product and the k-mode product facilitate decomposition models for tensors, by allowing "lower rank approximations" of tensors similar to the matrix versions. However, only the t-product will allow us to construct a linear model using tensors, while the k-mode product is meant to construct multilinear models.

CHAPTER 3

TENSOR DECOMPOSITIONS

In this chapter, we describe and contrast notable tensor decompositions. These decomposition models represent the bulk of the tensor methodology used in facial recognition, data-mining, and statistical analysis of image populations.

Tensor decompositions have been predominantly multilinear, and we start with these more traditional decompositions. In Section 3.1 we discuss the CP decomposition, which introduced the notion of a tensor rank. In Section 3.2 we discuss the more general Tucker decomposition, which encompasses CP as a special case. In Section 3.3 and Section 3.4, we discuss two matrix-based models that actually have intimate connections with the Tucker model. Finally, in Section 3.5, we discuss Tensor Singular Value Decomposition (T-SVD) [27], a novel method based on the t-product, as well as related Tensor approximation schemes. Whereas the family of Tucker models decomposes a higher-order tensor into a higher-order core and factor matrices for each mode, the T-SVD decomposes into multiple higher-order tensors.

3.1 CP

The CP decomposition stemmed independently from psychometrics [13] and chemometrics [10], where the same method was separately named Canonical Decomposition (CANDECOMP) and Parallel Factors (PARAFAC). We first describe the related concept of ten-

sor rank.

A K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ is called rank-1 if it can be expressed as an outer product of K vectors. It is called rank-r if it can be expressed as a sum of r rank-1 K-tensors:

$$\mathcal{X} = \sum_{\ell=1}^{r} v_{1\ell} \circ v_{2\ell} \ldots \circ v_{K\ell}, \text{ where } v_{k\ell} \in \mathbb{R}^{n_k}, 1 \leq \ell \leq r, 1 \leq k \leq K.$$

For matrices, the well-known Eckhart Young Theorem provides the existence and form of an optimal lower-rank approximation. However, this type of result has been shown not to generalize to K-tensors for $K \geq 3$ [29].

Fortunately, the CP decomposition provides an approximation of $\mathcal{X}$ using a rank-r tensor $\hat{\mathcal{X}}$, where r is given a priori. The goal is then to construct a rank-r tensor that minimizes the Frobenius norm of the difference between $\mathcal{X}$ and $\hat{\mathcal{X}}$:

$$\begin{aligned}
&\min_{\text{all } u_{k\ell}} \|\mathcal{X} - \hat{\mathcal{X}}\|_F, \text{ where} \\
&\quad \hat{\mathcal{X}} = \sum_{\ell=1}^{r} v_{1\ell} \circ v_{2\ell} \ldots \circ v_{K\ell} && (3.1) \\
&\quad = \sum_{\ell=1}^{r} \lambda_\ell \cdot u_{1\ell} \circ u_{2\ell} \ldots \circ u_{K\ell}, u_{k\ell} = \frac{v_{k\ell}}{\|v_{k\ell}\|} && (3.2) \\
&\quad = \Lambda \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K, \text{ where} && (3.3)
\end{aligned}$$

$$U_k = \begin{bmatrix} u_{k1} & u_{k2} & \ldots & u_{kr} \end{bmatrix} \in \mathbb{R}^{n_k \times r}, 1 \leq k \leq K,$$

and

$$\Lambda \in \mathbb{R}^{r \times r \times r}$$

is a 3-tensor that contains the lambdas on the super-diagonal and 0 everywhere else, as seen in Figure 3.1.

The equivalence of lines (3.1) and (3.2) are due to the fact that each $u_{k\ell}$ vector is $v_{k\ell}$ normalized by its norm with the norm information stored in the λ_r's. Furthermore, we can store the u_{ir} vectors as a factor matrix U_k for each $k = 1, \ldots, K$ [32], leading to the form in line (3.3). Note that here the U_k matrices are not orthogonal. This relationship is illustrated for a 3-tensor below in Figure 3.1.

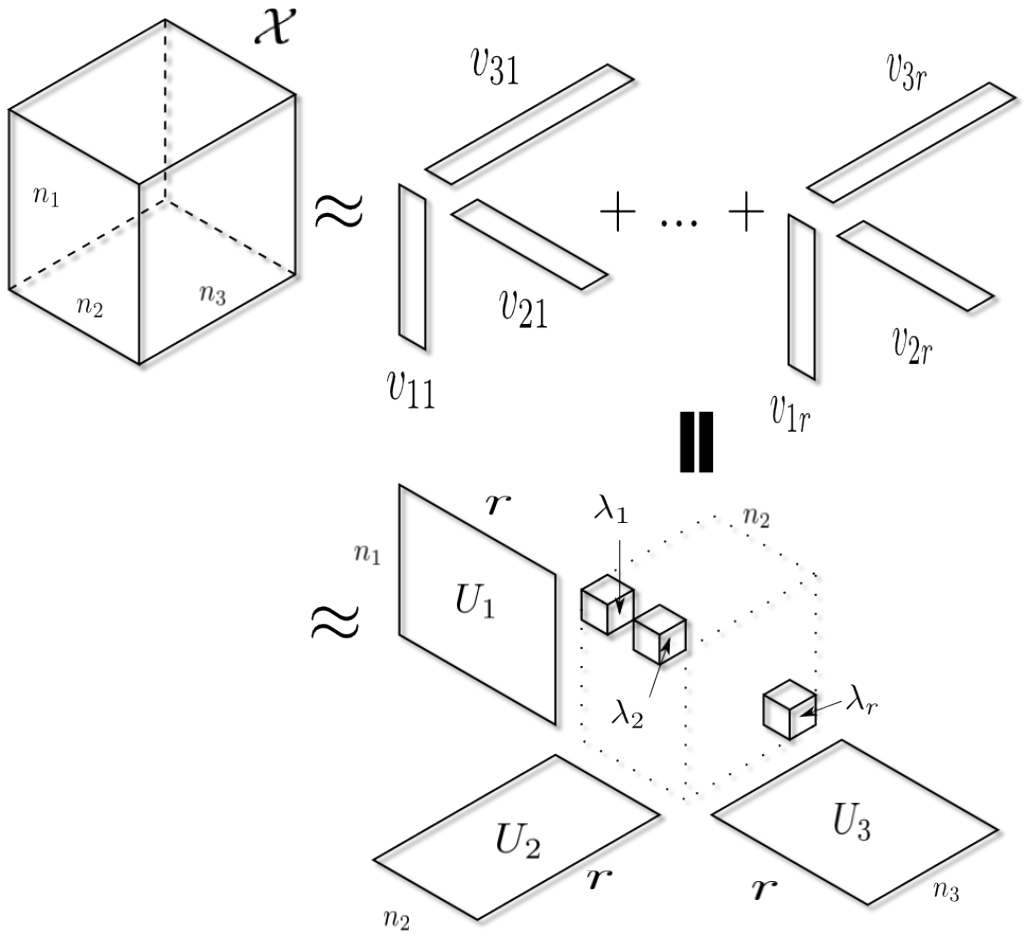


Figure 3.1: CP Decomposition for a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. The first part shows a representation using a sum of rank-1 tensors. The second part shows a representation using factor matrices.

A large body of literature has been devoted to obtaining $\hat{\mathcal{X}}$ with various possible solutions ranging from iterative to closed-formed [33, 32, 11, 51]. Some methods also exist to estimate r and $\hat{\mathcal{X}}$ simultaneously [11].

We consider the basic iterative procedure known as alternating least squares (ALS), as described in [28]. ALS for a 3-tensor proceeds with choosing some starting values for matrices U_1, U_2, U_3. Then at each iteration, we hold two of these matrices fixed and minimize $\|\mathcal{X} - \hat{\mathcal{X}}\|_F$ with respect to the third matrix, then repeat across the other two modes. For instance, if we hold U_2, U_3 constant then U_1 has a closed form best approximation:

$$\begin{aligned}
\text{Set } \hat{\mathcal{X}}_{(1)} &= U_1 \cdot \Lambda_{(1)} \cdot (U_3 \otimes U_2)^T \\
\text{Setting } \frac{\partial(\|\mathcal{X} - \hat{\mathcal{X}}\|_F)}{\partial U_1} &= 0 \Rightarrow \\
\hat{U}_1 &= \hat{\mathcal{X}}_{(1)} \cdot [(U_3 \otimes U_2)^T]^\dagger \\
&= \hat{\mathcal{X}}_{(1)} \cdot (U_3 \odot U_2) \cdot (U_2^T U_2 * U_3^T U_3)^\dagger,
\end{aligned}$$

where † denotes the Moore-Penrose Pseudoinverse, This process is then repeated for each of the two other modes to find $\hat{U}_2$ and $\hat{U}_3$. The algorithm exploits connections between the Kronecker and both the Khatri-Rao ($\odot$) and Hamadard ($*$) products of matrices [28, 55]. A usual convergence criterion is when there is a small enough difference between successive $\|\mathcal{X} - \hat{\mathcal{X}}\|$, but there is no guarantee that CP using ALS will convergence to a global minimum. This procedure is provided in `rTensor` [36] for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$.

Algorithm 2 CP using Alternating Least Squares (CP-ALS)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, and desired rank r.
initialize: $U_1, \ldots, U_K$ random matrices
while *Not Converged* **do**
  for $k = 1, \ldots, K$ **do**
    $H \leftarrow U_1^T U_1 * U_2^T U_2 * \ldots * U_{k-1}^T U_{k-1} * U_{k+1}^T U_{k+1} \ldots * U_K^T U_K$
    $U_k \leftarrow \mathcal{X}_{(k)} \cdot (U_K \odot U_{K-1} \odot \ldots \odot U_{k+1} \odot U_{k-1} \ldots \odot U_1) \cdot H^{\dagger}$
    $[\lambda_1, \ldots, \lambda_r] \leftarrow$ norms of columns of U_k
  end
end
output : scalars $\lambda_1, \ldots, \lambda_r$, factor matrices $U_1, \ldots, U_K$, each $U_k \in \mathbb{R}^{n_k \times r}$

3.2 Tucker

The Tucker decomposition [53, 32] is still based on the idea of obtaining the best approximation of $\mathcal{X}$, but relaxing the constraint that $\hat{\mathcal{X}}$ must be expressed as a sum of r rank-1 tensors. Instead, the Tucker decomposition constructs $\hat{\mathcal{X}}$ to approximate $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ using a reduced core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ and K factor matrices, each of rank $r_k \leq n_k$, $k = 1, \ldots, K$:

$$\hat{\mathcal{X}} = \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K.$$

If this looks similar to the CP, it is because the CP decomposition can be seen as the Tucker decomposition with all the ranks equal, (i.e. $r = r_1 = \ldots = r_K$) [28]. The general Tucker decomposition does not give a unique solution, however, so certain constraints must be placed on the factor matrices to ensure uniqueness. In this paper, we focus on Higher Order Orthogonal Iteration (HOOI) [33], which constrains the factor matrices to

be orthogonal, thus resulting in a unique solution.

Before we demonstrate HOOI, we first discuss the very much related higher-order singular value decomposition (HOSVD) provided by the seminal paper by Lathauwer et. al. [33]. HOSVD decomposes a K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ as follows:

$$\mathcal{X} = \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K,$$

where each square matrix $U_k \in \mathbb{R}^{n_k \times n_k}$ is orthogonal and the core tensor $\mathcal{G} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ has the special property that for any k, $1 \leq k \leq K$, it is

- all-orthgonal: $\langle \mathcal{G}_{i_k=\alpha}, \mathcal{G}_{i_k=\beta} \rangle = 0$ for any $\alpha \neq \beta$, and
- ordered: $\|\mathcal{G}_{i_k=1}\|_F \geq \|\mathcal{G}_{i_k=2}\|_F \geq \ldots \|\mathcal{G}_{i_k=n_k}\|_F$.

For 3-tensors, all-orthogonality of $\mathcal{G}$ means that for any of the three modes, any two matrix slices with different indices along that mode has an inner-product of 0. While one might expect the core tensor $\mathcal{G}$ to have some sort of diagonal structure, that is not the case here.

The corresponding algorithm to compute the HOSVD illustrates its crucial connection between the k-mode unfolding: for each k-mode unfolding, perform a matrix SVD for $\mathcal{X}_{(k)}$ so that $\mathcal{X}_{(k)} = U_k \Sigma_k V_k^T$. Now we can simply define $\mathcal{G} := \mathcal{X} \times_1 U_1^T \times_2 U_2^T \ldots \times_K U_K^T$, then

$$\begin{aligned}
\mathcal{G}_{(k)} &= U_n^T \cdot \mathcal{X}_{(k)} \cdot (U_K^T \otimes \ldots \otimes U_{k+1}^T \otimes U_{k-1}^T \otimes \ldots \otimes U_1^T)^T \\
\Rightarrow \mathcal{X}_{(k)} &= U_k \cdot \mathcal{G}_{(k)} \cdot (U_K \otimes \ldots \otimes U_{k+1} \otimes U_{k-1} \otimes \ldots \otimes U_1)^T \\
\Rightarrow \mathcal{X} &= \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K,
\end{aligned}$$

giving us the HOSVD. The following algorithm is implemented in `rTensor` [36].

Algorithm 3 Higher-Order Singular Value Decomposition (HOSVD)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$.
for $k = 1, \ldots, K$ **do**
  $U_k \leftarrow$ left orthogonal matrix of the SVD of $\mathcal{X}_{(k)}$
end
$\mathcal{G} \leftarrow \mathcal{X} \times_1 U_1^T \times_2 U_2^T \times_3 \ldots \times_K U_K^T$
output : core tensor $\mathcal{G} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, orthgonal factor matrices $U_1, \ldots, U_K$, each $U_k \in \mathbb{R}^{n_k \times n_k}$

The key concept here that allows us to obtain a Tucker decomposition from HOSVD is that if we truncate each U_k to its first r_k columns for each $1 \leq k \leq K$, then we would end up with an approximation of $\mathcal{X}$. Successive iterations of this alternating truncation and SVD for all the modes will give us a locally optimized approximation $\hat{\mathcal{X}}$. This is known as the higher-order orthogonal iteration (HOOI) [33] and provides an iterative scheme to find the Tucker decomposition with orthogonality constraints.

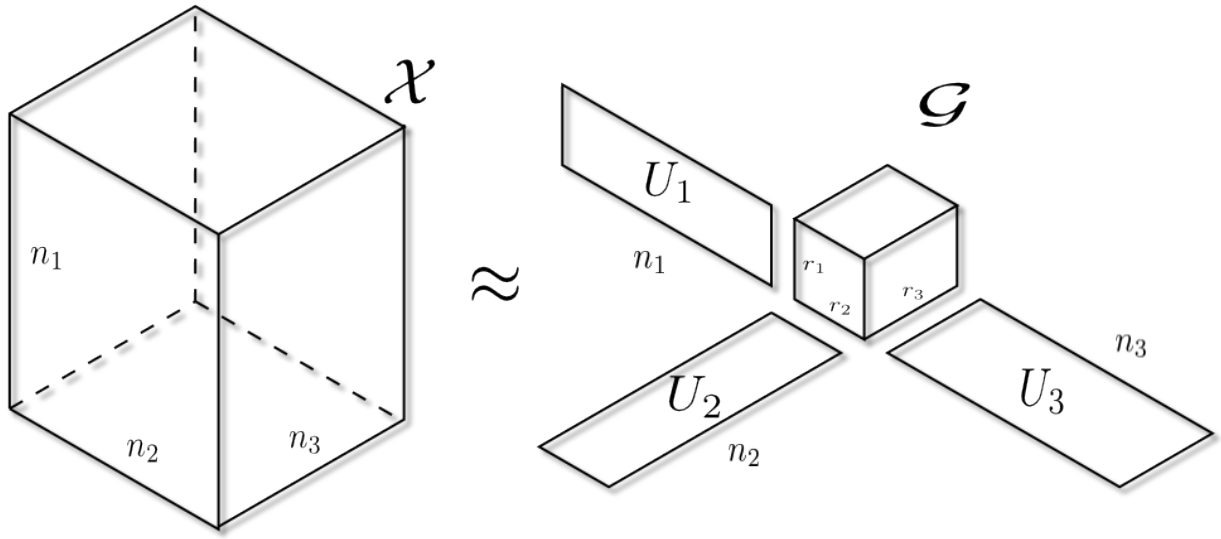
Algorithm 4 Orthogonal Tucker Alternating Least Squares (HOOI)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, and desired ranks $r_1, \ldots, r_K$.
initialize: $U_1, \ldots, U_K$ via HOSVD
while *Not Converged* **do**
  for $k = 1, \ldots, K$ **do**
    $\mathcal{Y} \leftarrow \mathcal{X} \times_1 U_1^T \times_2 \ldots \times_{k-1} U_{k-1}^T \times_{k+1} U_{k+1}^T \ldots \times_K U_K^T$
    $U_k \leftarrow r_k$ leading singular vectors of $\mathcal{Y}_{(k)}$
  end
end
$\mathcal{G} \leftarrow \mathcal{X} \times_1 U_1^T \times_2 \ldots \times_K U_K^T$
output : core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$, factor matrices with orthogonal columns $U_1, \ldots, U_K$, each $U_k \in \mathbb{R}^{n_k \times r_k}$

Note that in many modified versions of HOOI, the truncated HOSVD only serves as an initial value. Successive iterations do not require a full SVD at each mode for the matricized tensor, but only some scheme to minimize

$$\|\mathcal{X} - \mathcal{G} \times_1 U_1 \times_2 \ldots \times_{k-1} U_{k-1} \times_{k+1} U_{k+1} \ldots \times_K U_K\|_F,$$

for $1 \leq k \leq K$. See [28, 55, 33, 32].



$$\mathcal{X} \approx \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 U_3$$

Figure 3.2: Tucker Decomposition for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, resulting in factor matrices with orthogonal columns $U_k \in \mathbb{R}^{n_k \times r_k}, k = 1, 2, 3$ and an all-orthgonal core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$.

By allowing the specification of the ranks $r_1, \ldots, r_K$, Tucker offers more flexibility than CP in the amount of truncation along each mode of $\mathcal{X}$. Furthermore, with HOOI, we are also given orthogonal factor matrices. These properties makes HOOI a much more attractive statistical model than CP, and have lead to a number of statistical tensor regression methods, as we will see in Chapter 4.

It is important to note that in the absence of truncation, the Tucker decomposition (with orthogonality constraints) can be viewed as just the HOSVD and therefore does not require an iterative algorithm to obtain the factor matrices U_k's. The core tensor $\mathcal{G}$ is therefore not

compressed and will still be of size $n_1 \times n_2 \times \ldots \times n_K$. Since tensor decompositions are often used to compress large tensors, this does not seem very practical. It does, however, give us a way to compare the structural changes across many different methods, as we will see in the following sections.

3.3 GLRAM, MPCA, & 2dPCA

In this section, we discuss several statistical models that are unified under the Tucker framework. While some of these models explicitly use tensor notation and methodology, others use a series of matrices as inputs.

The Generalized Low Rank Approximate of Matrices (GLRAM) [59] belongs in the latter category. For a series of matrices of the same size, $M_1, \ldots, M_{n_3} \in \mathbb{R}^{n_1 \times n_2}$, GLRAM constructs orthogonal matrices $L \in \mathbb{R}^{n_1 \times r_1}, R \in \mathbb{R}^{n_2 \times r_2}$ and a series of core matrices $G_j \in \mathbb{R}^{r_1 \times r_2}$ to minimize the quantity $\sum_{j=1}^{n_3} \|M_j - L \cdot G_j \cdot R^T\|_F^2$. The parameters r_1 and r_2 would also need to be given a priori.

The series of images GLRAM takes as input can be restructured into a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, where $\mathcal{X}[:, :, j] = M_j$. This has been done in [52], where it is shown that when structured in this way, GLRAM becomes a special case of the Tucker decomposition with orthogonality constraints. Essentially, GLRAM performs HOOI in 2 of the 3 modes, while leaving the third mode uncompressed. To make the comparison between the matrix and tensor settings more explicit, We present the same GLRAM algorithm using both the

matrix notation and the tensor notation below:

Algorithm 5 Generalized Low Rank Approximation of Matrices using matrix notation

input : Matrices $M_1, \ldots, M_{n_3}$, ranks r_1, r_2.
initialize: L random matrix
while *Not Converged* **do**
  Form $C_R = \sum_{j=1}^{n_3} M_j^T \cdot L \cdot L^T \cdot M_j$
  $R \leftarrow r_2$ eigenvectors corresponding to the r_2 largest eigenvalues of C_R
  Form $C_L = \sum_{j=1}^{n_3} M_j \cdot R \cdot R^T \cdot M_j^T$
  $L \leftarrow r_1$ eigenvectors corresponding to the r_1 largest eigenvalues of C_L
end
for $j = 1, \ldots, n_3$ **do**
  $V_j \leftarrow L^T \cdot M_j \cdot R$
end
output : $L, R, V_1, \ldots, V_{n_3}$

Algorithm 6 GLRAM using tensor notation

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, ranks r_1, r_2.
while *Not Converged* **do**
  Form $C_R = \mathcal{X}_{(1)} \cdot \mathcal{X}_{(1)}^T$
  $R \leftarrow r_2$ eigenvectors corresponding to the r_2 largest eigenvalues of C_R
  $\mathcal{X} \leftarrow \mathcal{X} \times_2 R^T$
  Form $C_L = \mathcal{X}_{(2)} \cdot \mathcal{X}_{(2)}^T$
  $L \leftarrow r_1$ eigenvectors corresponding to the r_1 largest eigenvalues of C_L
  $\mathcal{X} \leftarrow \mathcal{X} \times_1 L^T$
end
for $j = 1, \ldots, n_3$ **do**
  $V_j \leftarrow L^T \cdot \mathcal{X}[:, :, j] \cdot R$
end
output : $L, R, V_1, \ldots, V_{n_3}$

One can see that during GLRAM, we are essentially performing HOOI over only the first two modes, since SVD of $\mathcal{X}_{(i)}$ gives the same left eigenvectors as the eigenvalue decomposition of $\mathcal{X}_{(i)} \cdot \mathcal{X}_{(i)}^T$, $i = 1, 2$.

When applied to a series of K-tensors for $K \geq 3$, this technique, is called Multilinear Principal Component Analysis (MPCA) [38]. MPCA is also a special case of the general Tucker decomposition for K-tensors, compressing on $K - 1$ modes and leaving one mode uncompressed. Hence GLRAM is a special case of MPCA for 3-tensors. Notationally, MPCA is equivalent to HOOI with $U_K = I_{n_k}$, which means that GLRAM is equivalent to HOOI with $U_3 = I_{n_3}$.

Yet another related technique is called 2-dimensional Principal Component Analysis (2dPCA) [58, 61], and that is also shown to be a special case of GLRAM [52], where two of the three modes are uncompressed (i.e. $U_2 = I_{n_2}, U_3 = I_{n_3}$). All the decompositions mentioned above (HOOI, GLRAM, 2dPCA, and MPCA) are available in `rTensor` [36].

3.4 PVD

Recently proposed by [14], the Population Value Decomposition (PVD) provides a framework to construct population-level factor matrices for a series of images. We show in this section that PVD actually is a variant of GLRAM. This point was first made by Lock et al. in the rejoinder of the original PVD paper [14], and Crainiceanu et al. replied that PVD differs from Tucker (or specifically, GLRAM) in many ways. Most notably, the matrices

P and D do not have to be orthogonal and that the default PVD has a closed form solution. We first present PVD and the default algorithm suggested by the authors to construct the population matrices P and D, then examine the differences between PVD and GLRAM. Finally, we discuss how PVD might be cast in the tensor framework.

Like GLRAM and 2dPCA, PVD is a model designed for a series of matrices instead of a 3-tensor setup. Given a sample of images $M_1, \ldots, M_{n_3} \in \mathbb{R}^{n_1 \times n_2}$, and $2n_3 + 2$ parameters, PVD constructs population level matrices $P \in \mathbb{R}^{n_1 \times r_1}$ and $D \in \mathbb{R}^{n_2 \times r_2}$ such that $X_j = P \cdot V_j \cdot D + E_j$, where the $V_j \in \mathbb{R}^{r_1 \times r_2}$, $j = 1, \ldots, n_3$, are called the core matrices. In addition to the 2 parameters $r_i \leq n_i$, $i = 1, 2$, we also would need to choose $2n_3$ compression parameters, $l_1, \ldots, l_{n_3}, h_1, \ldots, h_{n_3}$, that will determine how much left and right truncation will occur for each of the n_3 matrices.

The PVD procedure starts with a separate SVD of each image, $M_j = U_j \Sigma_j W_j^T$, truncating (possibly differently for each image) the left and right eigenvectors to form $\tilde{U}_j$ and $\tilde{W}_j$. Then the one would stack the $\tilde{U}_j$'s column wise to form a big matrix U and do the same for $\tilde{W}_j$'s to form W. The final step is to conduct an eigenvalue decomposition of $U \cdot U^T$ and $W \cdot W^T$ to form the population level matrices P and D. In the end, each M_j has a projection

$$\hat{M}_j = P \cdot \underbrace{\{(P^T \cdot \tilde{U}_j) \cdot \Sigma_n^{(l_j, h_j)} \cdot (\tilde{W}_j \cdot D^T)\}}_{V_j} \cdot D,$$

where $\Sigma_j^{(l_j, h_j)}$ is the (l_j, h_j) left upper block of Σ_j.

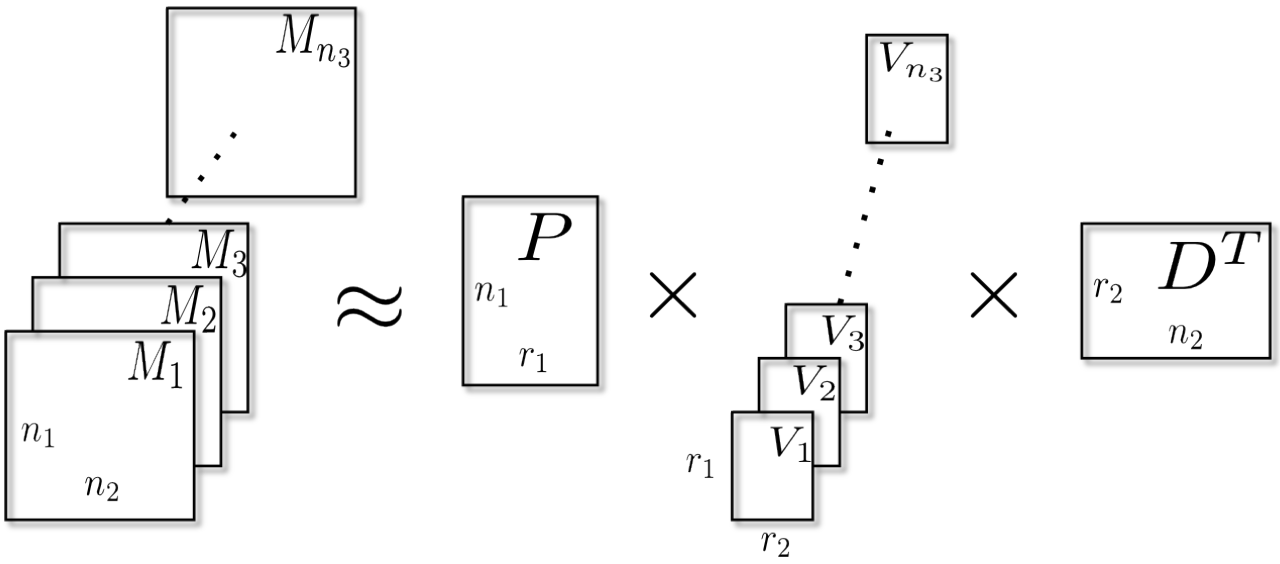


Figure 3.3: PVD of a series of images $M_1, \ldots, M_{n_3}$. Each M_j is approximated by $P \cdot V_j \cdot D^T$.

Unlike the usual algorithm needed to solve GLRAM, the algorithm to solve the default PVD is not iterative, although the computational cost of the model does scale up with the number of images, since each image requires a separate SVD. Furthermore, with a large n_3, the UU^T and WW^T matrices may be intractable for a full eigenvalue decomposition. We present the matrix version of the default PVD algorithm below, which is fully implemented in `rTensor` [36].

Algorithm 7 Default Population Value Decomposition (PVD)

input : Matrices $M_1, \ldots, M_{n_3}$, matrix-wise ranks $l_1, \ldots, l_{n_3}, h_1, \ldots, h_{n_3}$, final ranks r_1, r_2.

for $j = 1, \ldots, n_3$ **do**

Perform SVD to obtain $M_j = U_j \cdot \Sigma_j \cdot W_j^T$

$\tilde{U}_j \leftarrow$ left l_j columns of U_j

$\tilde{W}_j \leftarrow$ left h_j columns of W_j

end

Stack the $\tilde{U}_j$ column-wise to construct $U = \begin{bmatrix} \tilde{U}_1 & \ldots & \tilde{U}_{n_3} \end{bmatrix}$

and similarly stack the $\tilde{W}_j$ to form $W = \begin{bmatrix} \tilde{W}_1 & \ldots & \tilde{W}_{n_3} \end{bmatrix}$.

$P \leftarrow$ eigenvectors corresponding to the r_1 leading eigenvalues of $U \cdot U^T$

$D \leftarrow$ eigenvectors corresponding to the r_2 leading eigenvalues of $W \cdot W^T$

for $j = 1, \ldots, n_3$ **do**

$V_j \leftarrow P^T \cdot \tilde{U}_j \cdot \Sigma_j^{(l_j, r_j)} \cdot \tilde{W}_j^T D^T$

end

output : $P, D, V_1, \ldots, V_{n_3}$

Since both PVD and GLRAM are designed to work on a series of images M_j, a natural question is how do they compare with each other. Lock et al. gave an empirical comparison of both methods on the Orville Face dataset [12]. We found that in addition to these empirical similarities, if we were to compare both these methods under the case of **no truncation**, (i.e. no compression on any modes), then a striking similarity emerged.

First consider a GLRAM model for $\mathcal{X}$ with ranks $r_1 = n_1, r_2 = n_2$. This amounts to a HOOI for a 3-tensor with desired ranks $r_1 = n_1, r_2 = n_2, r_3 = n_3$, which means that the decomposition requires only 1 iteration since there is no compression. Furthermore, since

L and R are orthogonal, C_L and C_R take on much simpler forms:

$$\begin{aligned} C_L &= \sum_{j=1}^{n_3} M_j \cdot R \cdot R^T \cdot M_j^T \\ &= \sum_{j=1}^{n_3} M_j \cdot M_j^T \\ C_R &= \sum_{j=1}^{n_3} M_j^T \cdot L \cdot L^T \cdot M_j \\ &= \sum_{j=1}^{n_3} M_j^T \cdot M_j \end{aligned}$$

From here we set L and R to be equal to the left eigenvectors of of C_L and C_R.

Now consider PVD with $l_1 = \ldots = l_{n_3} = r_1 = n_1, h_1 = \ldots = h_{n_3} = r_2 = n_2$. After the n_3 SVD's, we have $U_j = M_j \Sigma_j^{-1} W_j$ and $W_j = M_j^T \Sigma_j^{-T} U_j$ for each $j = 1, \ldots, n_3$. Now we construct the large matrices U, W and set P, D equal to the left eigenvectors of $U \cdot U^T$, $W \cdot W^T$

respectively. However, it is easy to see that:

$$\begin{aligned}
U \cdot U^T &= \sum_{j=1}^{n_3} U_j \cdot U_j^T \\
&= \sum_{j=1}^{n_3} (M_j \cdot \Sigma_j^{-1} \cdot W_j) \cdot (W_j^T \cdot \Sigma_j^{-1} \cdot M_j^T) \\
&= \sum_{j=1}^{n_3} M_j \cdot \Sigma_j^{-2} \cdot M_j^T \\
W \cdot W^T &= \sum_{j=1}^{n_3} W_j \cdot W_j^T \\
&= \sum_{j=1}^{n_3} (M_j^T \cdot \Sigma_j^{-1} \cdot U_j) \cdot (U_j^T \cdot \Sigma_j^{-1} \cdot M_j) \\
&= \sum_{j=1}^{n_3} M_j^T \cdot \Sigma_j^{-2} \cdot M_j
\end{aligned}$$

From this we can make the following observations about GLRAM and PVD in the case without truncation:

- GLRAM has a closed form solution, requiring only 2 eigenvalue decompositions to solve.
- GLRAM reduces to computing the eigenvalue decomposition of

  $$\sum_{j=1}^{n_3} M_j \cdot M_j^T \text{ and } \sum_{j=1}^{n_3} M_j^T \cdot M_j.$$

  while PVD reduces to computing the eigenvalue decomposition of

  $$\sum_{j=1}^{n_3} M_j \cdot \Sigma_j^{-2} \cdot M_j^T \text{ and } \sum_{j=1}^{n_3} M_j^T \cdot \Sigma_j^{-2} \cdot M_j.$$

- PVD performs PCA on the weighted sum of the inner and outer matrix products of the images, with the weights being the singular values of each image. GLRAM, on the other hand, performs PCA on the simple sums. PVD requires n_3 additional SVD's to obtain the weights.

Naturally, PVD and GLRAM also differ in which step truncation is introduced. While PVD allows for a one-time truncation for each individual image and two final truncations for the inner and outer matrix products, GLRAM truncates at each iteration of the algorithm to minimize the objective function.

By casting GLRAM in the tensor framework, we see that GLRAM can be seen as a strict sub-model of the general Tucker decomposition. We do the same with PVD, and find that PVD does not fit exactly into the family of Tucker models due to the n_3 individual SVD's that are necessary to compute the weights. However, this suggests a hybrid model for higher-order tensors, one that would involve preprocessing each $K-1$-tensor to obtain weights before running a weighted Tucker decomposition on the full K-tensor.

3.5 T-SVD

Before we discuss the T-SVD, we first introduce the notion of the tensor transpose based on the t-product. Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then

$$\mathcal{X}^T := \text{fold}_{CS}(\begin{bmatrix} X_1^T \\ X_{n_3}^T \\ \vdots \\ X_2^T \end{bmatrix}), \text{ where } X_j = \mathcal{X}[:, :, j].$$

It is easily verified that $(\mathcal{X}^T)^T = \mathcal{X}$. Furthermore, let the identity tensor $\mathcal{I} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ be defined with $\mathcal{I}[:, :, 1] = I_{n_1}$, the matrix identity of size n_1, and the rest of $\mathcal{I}$ is set to 0. These two definitions then facilitate the notion of tensor orthogonality via the t-product:

$$\mathcal{Q} \in \mathbb{R}^{n_1 \times n_1 \times n_3} \text{ is orthogonal if and only if } \mathcal{Q} * \mathcal{Q}^T = \mathcal{Q}^T * \mathcal{Q} = \mathcal{I} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$$

As shown in [26], an orthogonal tensor preserves the Frobenius norm under the t-product. In other words, if $\mathcal{Q} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ and $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then $\|\mathcal{Q} * \mathcal{X}\|_F = \|\mathcal{X}\|_F$. We can now describe the T-SVD: let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then $\mathcal{X}$ admits a decomposition

$$\mathcal{X} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T,$$

where $\mathcal{U}, \mathcal{V}$ are orthogonal tensors of sizes $n_1 \times n_1 \times n_3$ and $n_2 \times n_2 \times n_3$ respectively, and $\mathcal{S}$ is of size $n_1 \times n_2 \times n_3$ and consists of diagonal matrices along the third mode. When $n_3 = 1$, then T-SVD reduces to the matrix SVD of $X \in \mathbb{R}^{n_1 \times n_2}$ [26]. This is a consequence of the fact that the t-product reduces to matrix multiplication when $n_3 = 1$.

The $\mathcal{S}$ tensor contains the **eigentubes** $\mathcal{S}[i, i, :], 1 \leq i \leq \tilde{n} := min(n_1, n_2)$, each of which is a vector of length n_3. Similar to the matrix eigenvalue counterparts, these eigentubes are ordered by the Frobenius norm:

$$\|\mathcal{S}[1, 1, :]\|_F \geq \|\mathcal{S}[2, 2, :]\|_F \geq \ldots \geq \|\mathcal{S}[\tilde{n}, \tilde{n}, :]\|_F.$$

As noted in Section 2.4, the discrete Fourier transform provides us with an efficient calculation of the T-SVD. Instead of calculating a SVD of the very large block circulant matrix circ(matvec($\mathcal{X}$)), we can simply perform the SVD calculations in the Fourier domain [22, 26]. The algorithm is presented below and available in `rTensor` [36].

Algorithm 8 Tensor Singular Value Decomposition (T-SVD)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{D}[i_1, i_2, :] = \text{fft}(\mathcal{X}[i_1, i_2, :])$
  end
end
for $j = 1, \ldots, n_3$ **do**
  Compute the SVD of the complex $\mathcal{D}[:, :, j]$ to yield $\mathcal{D}[:, :, j] = U_j \cdot \Sigma_j \cdot V_j^T$
  $\mathcal{U}[:, :, j] \leftarrow U_j$
  $\mathcal{V}[:, :, j] \leftarrow V_j$
  $\mathcal{S}[:, :, j] \leftarrow \Sigma_j$
end
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_1$ **do**
    $\mathcal{U}[i_1, i_2, :] = \text{ifft}(\mathcal{U}[i_1, i_2, :])$
  end
end
for $i_1 = 1, \ldots, n_2$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{V}[i_1, i_2, :] = \text{ifft}(\mathcal{V}[i_1, i_2, :])$
  end
end
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{S}[i_1, i_2, :] = \text{ifft}(\mathcal{S}[i_1, i_2, :])$
  end
end
output : Orthogonal tensors $\mathcal{U} \in \mathbb{R}^{n_1 \times n_1 \times n_3}, \mathcal{V} \in \mathbb{R}^{n_2 \times n_2 \times n_3}$ and $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ with diagonal slices along the third mode

Note that this computation is direct, and while it uses complex values, if $\mathcal{X}$ consists of real values, then $\mathcal{U}, \mathcal{V}$ and $\mathcal{S}$ are all real as well.

The ordering of the eigentubes gives a relatively straightforward to compute a "lower-

order" approximation of the original tensor: simply truncated at some $k \leq min(n_1, n_2)$, and compute

$$\tilde{\mathcal{X}} = \sum_{i=1}^{k} \mathcal{U}[:, i, :] * \mathcal{S}[i, i, :] * \mathcal{V}[:, i, :]^T.$$

This leads to the following compression strategy that uses a truncation index k.

Algorithm 9 Compression Strategy 1 based on T-SVD (T-Compress)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, truncation index $k \leq min(n_1, n_2)$

$[\mathcal{U}, \mathcal{V}, \mathcal{S}] \leftarrow$ T-SVD($\mathcal{X}$).

$\tilde{\mathcal{X}} = \sum_{i=1}^{k} \mathcal{U}[:, i, :] * \mathcal{S}[i, i, :] * \mathcal{V}[:, i, :]^T.$

output : $\tilde{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$

The authors do note, however, that this does not seem to be an effective method to compress a given tensor, since the rank of $\mathcal{X}$ is most likely above $min(n_1, n_2)$, which can be restrictive if either is small. They suggest an alternative method to compressing a 3-tensor that is based on the following property of the T-SVD.

The $n_1 \times n_2$ matrix formed by summing across all the faces of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ admits a matrix SVD $X = U \cdot \Sigma \cdot V^T$, where the matrices U, V^T, Σ can be formed by summing across across the faces of $\mathcal{U}, \mathcal{V}^T, \mathcal{S}$ respectively. In other words,

$$\sum_{j=1}^{n_3} \mathcal{X}[:, :, j] = (\sum_{j=1}^{n_3} \mathcal{U}[:, :, j]) \cdot (\sum_{j=1}^{n_3} \mathcal{S}[:, :, j]) \cdot (\sum_{j=1}^{n_3} \mathcal{V}^T[:, :, j]).$$

This property gives another compression possibility: truncate the matrix SVD of $\sum_{j=1}^{n_3} \mathcal{X}[:, :, k]$, and to use those truncated factor matrices to reconstruct $\hat{\mathcal{X}}$.

Algorithm 10 Compression Strategy 2 based on T-SVD (T-Compress2)

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}, r_1 \leq n_1, r_2 \leq n_2$

Let $X = \sum_{j=1}^{n_3} \mathcal{X}[:, :, j]$

Compute matrix SVD $X = U \cdot \Sigma \cdot V^T$

$\tilde{U}$ = first r_1 columns of U

$\tilde{V}$ = first r_2 columns of V

for $j = 1, \ldots, n_3$ **do**

$\quad \mathcal{T}[:, :, j] = \tilde{U}^T \cdot \mathcal{X}[:, :, j] \cdot \tilde{V}$

end

$\hat{\mathcal{X}} = \sum_{i=1}^{r_1} \sum_{\ell=1}^{r_2} \tilde{U}[:, i] \circ \tilde{V}[:, \ell] \circ \mathcal{T}[i, \ell, :]$

output : $\tilde{U} \in \mathbb{R}^{n_1 \times r_1}, \tilde{V} \in \mathbb{R}^{n_2 \times r_2}, \mathcal{T} \in \mathbb{R}^{r_1 \times r_2 \times n_3}$

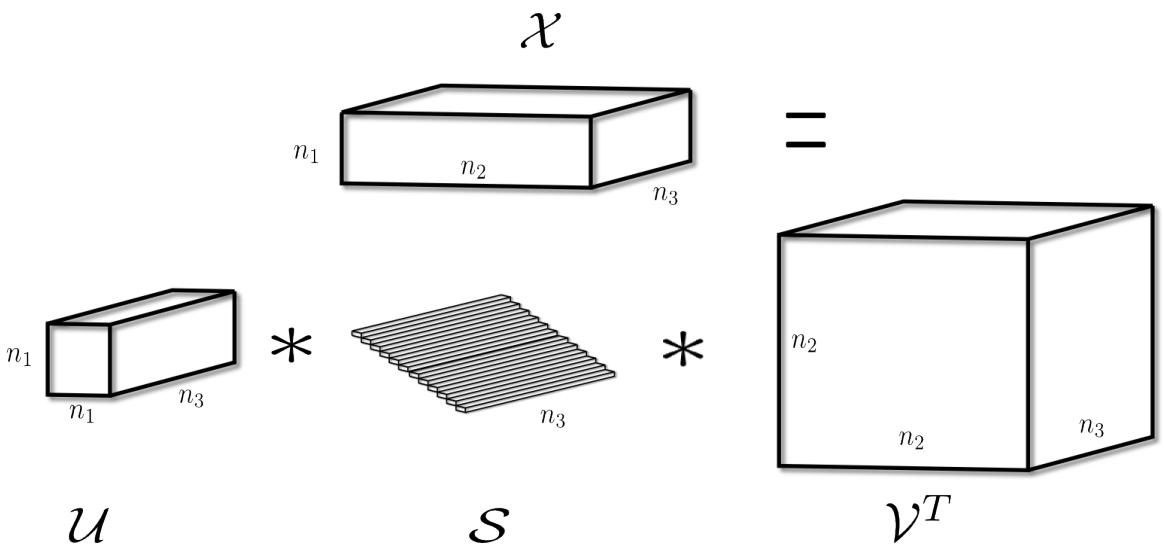


Figure 3.4: T-SVD for a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, resulting in two orthogonal tensors - $\mathcal{U} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ and $\mathcal{V} \in \mathbb{R}^{n_2 \times n_2 \times n_3}$ - and $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ has diagonal faces along n_3.

The order of the modes is important for the T-SVD and the two related compressions. This follows from the definition of the t-product and the fact that $\mathcal{A} * \mathcal{X}$ is a linear map in

the first two modes of $\mathcal{X}$. In particular, when $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ represents a series of images, the correct way to arrange $\mathcal{X}$ is to have each image as a slice along the second mode (i.e. $\mathcal{X}[:, j, :] = j^{th}$ image). This is discussed more at length in [27] and also in Chapter 5 when we present our tensor regression model.

CHAPTER 4

MULTILINEAR TENSOR REGRESSION

One of the primary motivations for tensor methodology is the use of tensor inputs for prediction and regression. This is especially true in fields where multilinear datasets are in abundance and there is a need for predictions and inference from tensor observations. In this chapter, we provide a brief review of the various tensor prediction models that have been developed so far. These models are multilinear since they depend - in one way or another - on the n-mode product.

In Section 4.1, we discuss a tensor regression model by [65] that predicts a univariate response using a higher-order array input. We also discuss a multilinear partial least squares model. In Section 4.2 we describe Generalized Linear Array Model, which also uses the n-mode product in its construction but swaps the parameters with the data in the multiplication. Finally, in Section 4.3, we describe the Multilinear Normal Distribution, which generalizes the matrix-variate Normal to K-tensors.

4.1 Tensor Regression via the Tucker

Zhou et al. developed a tensor regression model based on the Tucker decomposition [65]. Its parametrization involves a tensor core being the observed data and the parameters being the factor matrices. The response for the i^{th} observation y_i is univariate, which means that with replication, the output to the model is a vector. Hence $\mathbf{y} = (y_1, \ldots, y_n)$ belongs to the multivariate exponential family.

The predictors are tensor-valued, and the number of modes in the set of predictors determine the number of parameter matrices that are needed to fit the model. For instance, if $\mathcal{X}_i \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is the observed tensor predictors for the i^{th} observation, then $\mathcal{X} = (\mathcal{X}_1, \ldots, \mathcal{X}_n)$ is the 4-tensor input to the model. This is then related to $\mathbf{y}$ via the vec operation:

$$g(\mathbf{y}) = \alpha + (\beta_4 \otimes \beta_3 \otimes \beta_2 \otimes \beta_1)^T \text{vec}(\mathcal{X}),$$

where g is the link function and α is the intercept. Zhou et al. also provided a mixed model representation extension to this basic model, as well as various link functions for responses that are non-normal. We note here that the fact that $\mathbf{y}$ is a vector means that eventually, the output from the model must be some inner product between the parameters and the data. For responses that are vector or matrix variate, for instance, the output will be matrices or even higher-order tensors, and it is unclear how this model will generalize to accommodate for this.

In cases where the output is a tensor, one idea is to model a latent core tensor associated with both the input and the output, then learn the observed factor matrices through the data [63]. Essentially, perform simultaneous HOOI on both the input $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ and output $\mathcal{Y} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ with the constraint that one of the factor matrices must be the same across $\mathcal{X}$ and $\mathcal{Y}$. This factor matrix is the latent matrix that represents the shared attributes between $\mathcal{X}$ and $\mathcal{Y}$. This is known as the Multilinear Partial Least Squares (MPLS) model.

4.2 Generalized Linear Array Model

An alternative view on the multilinear regression problem is given by the Generalized Linear Array Model [15]. Currie et. al. proposed for GLAM for multidimensional smoothing. One interesting difference between GLAM and the other tensor models is that the parametrization casts the parameters as the core tensor and the data being the factor matrices.

The model is designed for a K-tensor output $\mathcal{Y} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ and model matrix with a special kronecker design:

$$X = X_1 \otimes X_2 \otimes \ldots \otimes X_K.$$

Here each of the X_i are known as marginal model matrices. The parameters are then a vector that have an induced tensor structure when folded.

This is naturally suited for spline bases or functional design since the data is evenly spaced out and admits a Kronecker product form easily. It also makes an important point that in the case where the output is a vector and not a matrix, then we end up with the same model as GLM. Currie et. al. also provide some compelling evidence suggesting that the tensor framework is numerically more efficient than the matrix framework. One question that remains to be answered is if GLAM can be effectively applied to cases where $\mathcal{X}$ is not a result of Kronecker design.

4.3 Multilinear Normal Distribution

The multilinear normal distributions have been introduced and used by [46, 24], extending the matrix (bilinear) normal distribution. These distributions are defined such that if the tensor is vectorized, then it would follow a multivariate normal distribution.

Defintion 14. *$X \in \mathbb{R}^{M\times N}$ has matrix normal distribution [21] with parameters (μ, Σ, Φ) iff $vec(X) \sim N_{MN}(vec(\mu), \Sigma \otimes \Phi)$, where the latter is a multivariate normal distribution [41].*

The general K-tensor case is extended using an additional covariance matrix for each additional mode:

Defintion 15. *$\mathcal{X} \in \mathbb{R}^{n_1\times n_2\times\ldots\times n_K}$ has a **multilinear normal distribution** with parameters $(\mu, \Sigma_1, \ldots, \Sigma_K)$ iff*

$$vec(\mathcal{X}) \sim N_{n_1\ldots n_K}(vec(\mu), \Sigma_1 \otimes \Sigma_2 \otimes \ldots \otimes \Sigma_K).$$

The density is given by:

$$f_{\mathcal{X}}(x) = (2\pi)^{N^*/2}\Big(\prod_{i=1}^{K} |\Sigma_i|^{N^*/(2N_i)}\Big) \exp\{-\frac{1}{2}(x-\mu)^T(\Sigma_1 \otimes \Sigma_2 \otimes \ldots \otimes \Sigma_K)^{-1}(x-\mu)\},$$

where $N^* = \prod_{i=1}^{K} N_i$. Σ_i is the covariance matrix associated with the i^{th} mode, $i = 1, \ldots, K$.

As a consequence of this definition, the covariance between element $(i_1, i_2, \ldots i_K)$ and element $(j_1, j_2, \ldots j_K)$ is a product of the corresponding elements of these covariance matrices:

$$cov(\mathcal{X}_{(i_1,i_2,\ldots i_K)}, \mathcal{X}_{(j_1,j_2,\ldots j_K)}) = (\Sigma_1)_{i_1,j_1}(\Sigma_2)_{i_2,j_2}\ldots(\Sigma_K)_{i_K,j_K}.$$

This definition of the multilinear normal allows for fairly parsimonious modeling of data, as well as a structured way to think about the correlations between arbitrary indices of the tensor. Another main advantage to this definition is the tractability of inversion offered by the Kronecker products. Namely, $(\Sigma_1 \otimes \Sigma_2 \otimes \ldots \otimes \Sigma_K)^{-1} = \Sigma_1^{-1} \otimes \Sigma_2^{-1} \otimes \ldots \otimes \Sigma_K^{-1}$. Another advantage of this definition is that the marginal and conditional distributions are also multilinear normal.

The maximum likelihood equations for the multilinear normal are given in [46]. $\hat{\mu}$ is the empirical average, but $\hat{\Sigma}_i$'s are nor identifiable unless we place the restriction that $K-1$ of the $(\Sigma_j)_{j,j}$ are equal to 1. Even with this restriction, however, the likelihood equations do not have an explicit solution and thus require an iterative estimation scheme such as the flip-flop algorithm [16] to converge.

Hoff espouses the same multilinear normal distribution in [24], motivating it from the concept of separable covariance arrays. Hoff provides a Bayesian estimation procedure for μ and $\Sigma_1, \ldots, \Sigma_K$, but also places the same identification constraint given by [46]. Hoff also shows that the notion of conditional distributions carry over from the multivariate setting in a very similar way for multilinear normals.

CHAPTER 5

TENSOR LINEAR MODEL

In this chapter, we introduce a novel 3-tensor regression model called Tensor Linear Model (TLM). In contrast to the multilinear models we surveyed in Chapter 4, TLM features many classical properties of the ordinary least squares, which serves as the foundation of many other nonlinear models for matrices. The derivation of TLM estimators and their properties are very similar to the matrix counterparts, further illustrating the relationship between the *t*-product and matrix multiplication.

We begin with the model setup in Section 5.1, then review the relevant 3-tensor operations and properties based on the *t*-product in Section 5.2. We then provide both the least squares and ridge estimators for TLM in Section 5.3. Finally, we provide algorithms to produce these estimates efficiently in Section 5.4 and a discussion for future directions in Section 5.5.

5.1 Model Setup

TLM is based on the novel multiplication between 3-tensors (the *t*-product) defined by Kilmer et al. in the paper "Factorization Strategies for Third-Order Tensors" [27], described in Section 2.4. TLM can be regarded as a multivariate response version of the Ordinary Least Squares (OLS) model. In fact, TLM contains OLS as a specific case in the univariate response setting as a consequence of the *t*-product. We believe that TLM is especially fitting for multivariate data that is observed over time, such as functional data

or vector auto-regressive processes.

First consider the response of the model. Let $\mathbf{y}_i \in \mathbb{R}^t$ be a vector-variate response corresponding to the i^{th} observation, $1 \leq i \leq n$. For each i, the k^{th} scalar of $\mathbf{y}_i$ represents the the k^{th} position of the variable, while the length of $\mathbf{y}_i$, denoted t, denotes the total number of positions in the i^{th} observation. For instance, in a functional dataset, t would represent the number of time points at which the function was measured.

The responses across all n individuals in the sample then can be represented as a matrix resulting from stacking the $\mathbf{y}_i$'s row-wise, i.e.

$$Y = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \cdots \\ \mathbf{y}_n^T \end{bmatrix} \in \mathbb{R}^{n \times t}.$$

Now consider the predictors/regressors to the model. Let $\mathbf{x}_{ij} \in \mathbb{R}^t$ be a vector-variate observation corresponding to the j^{th} predictor of the i^{th} individual, $1 \leq i \leq n$, $1 \leq j \leq p$. We represent the entire set of regressors as a 3-tensor, with the first mode representing the individuals, the second mode representing the number of covariates, and the third mode representing the length of of each vector-variate observation, i.e.

$$\mathcal{X} \in \mathbb{R}^{n \times p \times t}, \text{ where}$$

$$\mathcal{X}_{ijk} = \text{value for the } i^{th} \text{ observation, } j^{th} \text{ predictor, } k^{th} \text{ position.}$$

At this point, the second mode of Y and the second mode of $\mathcal{X}$ represent different things, so we need to "re-orient" Y such that it becomes a matrix in the 3-tensor space.

This "re-orientation" is coined the "*twist*" operation by Kilmer et al. in [26]. The basic idea is to consider Y as a matrix slice along the second mode or as a 3-tensor with $n_2 = 1$. To emphasize the idea that Y is now in the 3-tensor space, we denote it as:

$$\mathcal{Y} := twist(Y) \in \mathbb{R}^{n\times 1\times t}.$$

Defintion 16. *The **Tensor Linear Model (TLM)** parameterizes the linear relationship between the response $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$ and the regressors $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$ via a parameter slice, $\mathcal{B} \in \mathbb{R}^{p\times 1\times t}$:*

$$\mathcal{Y} = \mathcal{X} * \mathcal{B} + \mathcal{E},$$

where $\mathcal{E} \in \mathbb{R}^{n\times 1\times t}$ are the residuals to the model.

Note that in the case where $t = 1$ and assuming i.i.d. Gaussian residuals, TLM reduces to OLS.

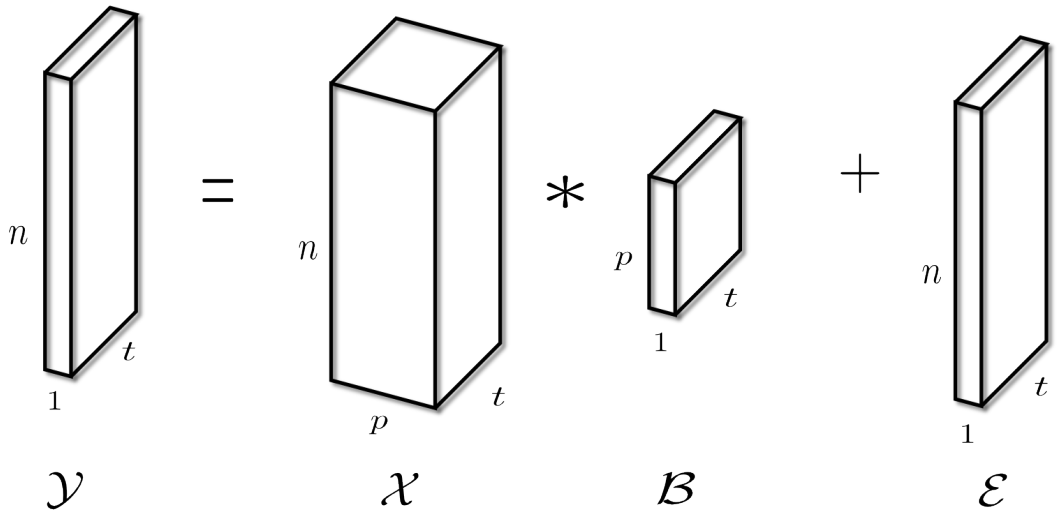


Figure 5.1: Tensor Linear Model (TLM) for $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$, $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, and $\mathcal{B} \in \mathbb{R}^{p\times 1\times t}$.

5.2 Review of 3-Tensor Operations and Properties

Before we discuss an estimate of the parameter $\mathcal{B} \in \mathbb{R}^{p\times 1\times t}$, we first review the notions of invertibility and orthgonality for 3-tensors. We also define the Moore-Penrose Pseudoinverse for 3-tensors and connect these notions to their matrix counterparts. Most of these properties are based off of Kilmer et al. [27], as we apply the t-product to the notions of least squares and prediction. It is important to note that these properties are a consequence of the fact that the t-product is a linear operation.

Defintion 17 (Kilmer et al. [27])**.** *$\mathcal{A} \in \mathbb{R}^{p\times p\times t}$ is **invertible** if there exists some $\mathcal{A}^{-1} \in \mathbb{R}^{p\times p\times t}$ such that*

$$\mathcal{A}^{-1} * \mathcal{A} = \mathcal{A} * \mathcal{A}^{-1} = \mathcal{I}_{ppt},$$

where $\mathcal{I}_{ppt} \in \mathbb{R}^{p\times p\times t}$ has I_n as the first slice and 0 everywhere else.

We now provide a necessary and sufficient condition for 3-tensor invertbility.

Lemma 5.2.1. *$\mathcal{A} \in \mathbb{R}^{p\times p\times t}$ is invertible if and only if $\mathbf{A} := \text{circ}(\text{matvec}(\mathcal{A})) \in \mathbb{R}^{pt\times pt}$ is invertible.*

Proof. We first prove the "if" direction:

Assume $\mathbf{A}$ is invertible, then there exists some matrix $\mathbf{B} \in \mathbb{R}^{pt\times pt}$ such that $\mathbf{AB} = \mathbf{BA} = I_{pt}$. Now since $\mathbf{A}$ is block circulant, then $\mathbf{B}$ is block circulant as well [43, 57]. Writing

$\mathbf{AB} = I_{pt}$ out, we then have:

$$\begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_t & B_{t-1} & \dots & B_2 \\ B_2 & B_1 & B_t & \dots & B_3 \\ B_3 & B_2 & B_1 & \dots & B_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_t & B_{t-1} & B_{t-2} & \dots & B_1 \end{bmatrix} = \begin{bmatrix} I_p & 0 & 0 & \dots & 0 \\ 0 & I_p & 0 & \dots & 0 \\ 0 & 0 & I_p & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I_p \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_t \end{bmatrix} = \begin{bmatrix} I_p \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\Rightarrow \mathcal{A} * \mathcal{B} = \mathcal{I}_{ppt} \text{ by the definition of } *, \text{ where } \mathcal{B} := \text{fold}_{CS}(\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_t \end{bmatrix}, 2, p \times p \times t).$$

Similarly, writing out $\mathbf{BA} = I_{pt}$, we have:

$$\begin{bmatrix} B_1 & B_t & B_{t-1} & \dots & B_2 \\ B_2 & B_1 & B_t & \dots & B_3 \\ B_3 & B_2 & B_1 & \dots & B_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_t & B_{t-1} & B_{t-2} & \dots & B_1 \end{bmatrix} \cdot \begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} = \begin{bmatrix} I_p & 0 & 0 & \dots & 0 \\ 0 & I_p & 0 & \dots & 0 \\ 0 & 0 & I_p & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I_p \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} B_1 & B_t & B_{t-1} & \dots & B_2 \\ B_2 & B_1 & B_t & \dots & B_3 \\ B_3 & B_2 & B_1 & \dots & B_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_t & B_{t-1} & B_{t-2} & \dots & B_1 \end{bmatrix} \cdot \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ A_t \end{bmatrix} = \begin{bmatrix} I_p \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$\Rightarrow \mathcal{B} * \mathcal{A} = \mathcal{I}_{ppt}$. Hence $\mathcal{A}$ is invertible and its inverse is $\mathcal{B}$.

We now prove the "only if" direction:

Assume $\mathcal{A} * \mathcal{B} = \mathcal{I}_{ppt}$, then by the definition of $*$, we have:

$$\begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_t \end{bmatrix} = \begin{bmatrix} I_p \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Now consider the matrix product $\mathbf{C} = \mathbf{AB}$. Since the product between block circulant matrices must be block circulant, and we already have the first block column, we essentially

have

$$\begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} \begin{bmatrix} B_1 & B_t & B_{t-1} & \dots & B_2 \\ B_2 & B_1 & B_t & \dots & B_3 \\ B_3 & B_2 & B_1 & \dots & B_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_t & B_{t-1} & B_{t-2} & \dots & B_1 \end{bmatrix} = \begin{bmatrix} C_1 & C_t & C_{t-1} & \dots & C_2 \\ C_2 & C_1 & C_t & \dots & C_3 \\ C_3 & C_2 & C_1 & \dots & C_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_t & C_{t-1} & C_{t-2} & \dots & C_1 \end{bmatrix}$$

$$= \begin{bmatrix} I_p & 0 & 0 & \dots & 0 \\ 0 & I_p & 0 & \dots & 0 \\ 0 & 0 & I_p & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I_p \end{bmatrix}.$$

Similarly, we have $\mathbf{BA} = I_{pt}$ by $\mathcal{B} * \mathcal{A} = \mathcal{I}_{ppt}$, so $\mathbf{A}$ is invertible and its inverse is $\mathbf{B} = \text{circ}(\text{matvec}(\mathcal{B}))$. □

Recall the definitions of 3-tensor transpose and orthogonalilty.

Defintion 18 (Kilmer et al. [27])**.** *Let $\mathcal{A} \in \mathbb{R}^{n\times p\times t}$, then the **transpose** of $\mathcal{A}$ is*

$$\mathcal{A}^T := \text{fold}_{CS}(\begin{bmatrix} A_1^T \\ A_t^T \\ \vdots \\ A_2^T \end{bmatrix}, 2, p \times n \times t).$$

*$\mathcal{Q} \in \mathbb{R}^{n\times n\times t}$ is **orthogonal** if $\mathcal{Q}^T = \mathcal{Q}^{-1}$.*

We can now prove the second and third lemmas.

Lemma 5.2.2. *Let* $\mathcal{A} \in \mathbb{R}^{p\times n\times t}$ *and denote* $\mathbf{A} := \text{circ}(\text{matvec}(\mathcal{A})) \in \mathbb{R}^{pt\times nt}$. *Then* $\text{circ}(\text{matvec}(\mathcal{A}^T)) = \mathbf{A}^T$.

Proof.

$$\begin{aligned}
\text{circ}(\text{matvec}(\mathcal{A}^T)) &= \text{circ}(\begin{bmatrix} A_1^T \\ A_t^T \\ \dots \\ A_2^T \end{bmatrix}) \\
&= \begin{bmatrix} A_1^T & A_2^T & A_3^T & \dots & A_t^T \\ A_t^T & A_1^T & A_2^T & \dots & A_{t-1}^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_2^T & A_3^T & A_4^T & \dots & A_1^T \end{bmatrix} \\
&= \begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix}^T \\
&= \mathbf{A}^T
\end{aligned}$$

□

Lemma 5.2.3. *Let* $\mathcal{A} \in \mathbb{R}^{p\times n\times t}$, *and* $\mathcal{B} \in \mathbb{R}^{n\times p\times t}$. *Denote* $\mathbf{A} := \text{circ}(\text{matvec}(\mathcal{A})) \in \mathbb{R}^{pt\times nt}$ *and* $\mathbf{B} := \text{circ}(\text{matvec}(\mathcal{A})) \in \mathbb{R}^{nt\times pt}$, *then*

$$\mathbf{AB} = \text{circ}(\text{matvec}(\mathcal{A} * \mathcal{B})) \in \mathbb{R}^{pt\times pt}.$$

which is also block circulant.

Proof. We first note that the product of two block circulant matrices is also block circulant [43]. Since both **A** and **B** are block circulant, then that means that the product **AB** is block circulant and completely determined by the first p columns, which are

$$\begin{bmatrix} A_1 & A_t & A_{t-1} & \dots & A_2 \\ A_2 & A_1 & A_t & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t & A_{t-1} & A_{t-2} & \dots & A_1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_t \end{bmatrix} = \text{circ}(\text{matvec}(\mathcal{A}))\text{matvec}(\mathcal{B})$$

$$= \text{matvec}(\mathcal{A} * \mathcal{B}).$$

Now as circ(matvec($\mathcal{A} * \mathcal{B}$)) is also block circulant and hence completely determined by the first p columns of matvec($\mathcal{A} * \mathcal{B}$), we have our result. □

What about when $\mathcal{A}$ is not invertible? Recall that the Moore-Penrose Pseudoinverse [8] of a matrix $A \in \mathbb{R}^{n \times p}$ with rank p for $n > p$ is defined to be the unique matrix, denoted $A^\dagger := (A^T A)^{-1} A^T \in \mathbb{R}^{p \times n}$, satisfying all of the following:

1. $AA^\dagger A = A$,
2. $A^\dagger A A^\dagger = A^\dagger$,
3. $(AA^\dagger)^T = AA^\dagger$,
4. $(A^\dagger A)^T = A^\dagger A$.

We can now define a similar structure for a 3-tensor.

Defintion 19. *Let $\mathcal{A} \in \mathbb{R}^{n\times p\times t}$ be given, then define its **Moore-Penrose Pseudoinverse** to be*

$$\mathcal{A}^\dagger := \text{fold}_{CS}(\mathbf{A}^\dagger[:, 1:p], 2, n \times p \times t),$$

where $\mathbf{A}^\dagger$ is the matrix Moore-Penrose Pseudoinverse of $\mathbf{A} = \text{circ}(\text{matvec}(\mathcal{A}))$.

Lemma 5.2.4. *For any $\mathcal{A} \in \mathbb{R}^{n\times p\times t}$, $\mathcal{A}^\dagger$ exists and is unique. Furthermore, it satisfies the following conditions:*

1. $\mathcal{A} * \mathcal{A}^\dagger * \mathcal{A} = \mathcal{A}$,
2. $\mathcal{A}^\dagger * \mathcal{A} * \mathcal{A}^\dagger = \mathcal{A}^\dagger$,
3. $(\mathcal{A} * \mathcal{A}^\dagger)^T = \mathcal{A} * \mathcal{A}^\dagger$, *and*
4. $(\mathcal{A}^\dagger * \mathcal{A})^T = \mathcal{A}^\dagger * \mathcal{A}$.

Proof. For any $\mathcal{A} \in \mathbb{R}^{n\times p\times t}$, we have its corresponding $\mathbf{A} = \text{circ}(\text{matvec}(\mathcal{A}))$. The existence and uniqueness of $\mathcal{A}^\dagger$ is thus guaranteed by the fact that $\mathbf{A}^\dagger$ exists and is unique. We now prove that $\mathcal{A}^\dagger$ satisfies the four conditions.

1. Denote $\mathcal{J} := \mathcal{A} * \mathcal{A}^\dagger$, then by Lemma 5.2.3, we have $\text{circ}(\text{matvec}(\mathcal{J})) = \mathbf{A}\mathbf{A}^\dagger$. Now

$$\begin{aligned}
\text{circ}(\text{matvec}(\mathcal{A} * \mathcal{A}^\dagger * \mathcal{A})) &= \text{circ}(\text{matvec}(\mathcal{J} * \mathcal{A})) \\
&= \text{circ}(\text{matvec}(\mathcal{J}))\text{circ}(\text{matvec}(\mathcal{A})) \text{ by Lemma 5.2.3} \\
&= \mathbf{A}\mathbf{A}^\dagger\mathbf{A} \\
&= \mathbf{A} \text{ by property (1) of matrix Moore-Penrose Pseudoinverse.}
\end{aligned}$$

This means that $\mathcal{A} * \mathcal{A}^\dagger * \mathcal{A} = \text{fold}_{CS}(\mathbf{A}[:, 1:p], 2, p \times p \times t) = \mathcal{A}$ by definition.

2. Denote $\mathcal{K} := \mathcal{A}^\dagger * \mathcal{A}$, then we have $\text{circ}(\text{matvec}(\mathcal{K})) = \mathbf{A}^\dagger\mathbf{A}$. Now

$$\begin{aligned}\text{circ}(\text{matvec}(\mathcal{A}^\dagger * \mathcal{A} * \mathcal{A}^\dagger)) &= \text{circ}(\text{matvec}(\mathcal{K} * \mathcal{A}^\dagger)) \\ &= \text{circ}(\text{matvec}(\mathcal{K}))\text{circ}(\text{matvec}(\mathcal{A}^\dagger)) \\ &= \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger \\ &= \mathbf{A}^\dagger \text{ by property (2) of matrix Moore-Penrose Pseudoinverse.}\end{aligned}$$

This means that $\mathcal{A}^\dagger * \mathcal{A} * \mathcal{A}^\dagger = \text{fold}_{CS}(\mathbf{A}^\dagger[:, 1:p], 2, p \times p \times t) = \mathcal{A}^\dagger$ by definition.

3. We simply have to show that $\text{circ}(\text{matvec}((\mathcal{A} * \mathcal{A}^\dagger)^T)) = \mathbf{A}\mathbf{A}^\dagger$, and our result follows by definition.

$$\begin{aligned}\text{circ}(\text{matvec}((\mathcal{A} * \mathcal{A}^\dagger)^T)) &= \text{circ}(\text{matvec}((\mathcal{A}^\dagger)^T * \mathcal{A}^T)) \text{ by [27]} \\ &= \text{circ}(\text{matvec}((\mathcal{A}^\dagger)^T))\text{circ}(\text{matvec}(\mathcal{A}^T)) \text{ by Lemma 5.2.3} \\ &= (\mathbf{A}^\dagger)^T\mathbf{A}^T \text{ by Lemma 5.2.2} \\ &= (\mathbf{A}\mathbf{A}^\dagger)^T \\ &= \mathbf{A}\mathbf{A}^\dagger \text{ by property (3) of matrix Moore-Penrose Pseudoinverse.}\end{aligned}$$

4. This is similar to our proof for part (c).

$$\begin{aligned}\text{circ}(\text{matvec}((\mathcal{A}^\dagger * \mathcal{A})^T)) &= \text{circ}(\text{matvec}(\mathcal{A}^T * (\mathcal{A}^\dagger)^T)) \\ &= \text{circ}(\text{matvec}((\mathcal{A}^\dagger)^T))\text{circ}(\text{matvec}(\mathcal{A}^T)) \text{ by Lemma 5.2.3} \\ &= \mathbf{A}^T(\mathbf{A}^\dagger)^T \\ &= (\mathbf{A}^\dagger\mathbf{A})^T \\ &= \mathbf{A}^\dagger\mathbf{A} \text{ by property (4) of matrix Moore-Penrose Pseudoinverse.}\end{aligned}$$

□

We conclude this section with our main theorem, which we are now ready to prove. This theorem connects $\mathcal{X}^T * \mathcal{X}$ with its matrix counterpart $\mathbf{X^T X}$, show that the Moore-Penrose Pseudoinverse for 3-tensor $\mathcal{X}^T * \mathcal{X}$ is well-defined through the matrix version, and gives a sufficient condition for $\mathcal{X}^T * \mathcal{X}$ to be invertible.

Theorem 5.2.1. *Let* $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$ *and let* $\mathbf{X} = \text{circ}(\text{matvec}(\mathcal{X})) \in \mathbb{R}^{nt\times pt}$. *Denote*

$$C_{\mathcal{X}} := \mathcal{X}^T * \mathcal{X} \in \mathbb{R}^{p\times p\times t}$$

and denote

$$\mathbf{C_X} = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{pt\times pt}.$$

Then

1. $\mathbf{C_X} = \text{circ}(\text{matvec}(C_{\mathcal{X}}))$.
2. $\mathbf{C_X}^\dagger = (F_p \otimes I_t)\text{diag}(D_1^\dagger, D_2^\dagger, \ldots, D_t^\dagger)(F_p^* \otimes I_t)$, *where* $D_j \in \mathbb{C}^{p\times p}$, $1 \le j \le t$.
3. *The Moore-Penrose Pseudoinverse of* $C_{\mathcal{X}}$ *is*

$$C_{\mathcal{X}}^\dagger = \text{fold}_{CS}(\mathbf{C_X}^\dagger[:, 1:p], 2, p\times p\times t),$$

and it satisfies the following conditions:

(a) $C_{\mathcal{X}} * C_{\mathcal{X}}^\dagger * C_{\mathcal{X}} = C_{\mathcal{X}}$,

(b) $C_{\mathcal{X}}^\dagger * C_{\mathcal{X}} * C_{\mathcal{X}}^\dagger = C_{\mathcal{X}}^\dagger$,

(c) $(C_{\mathcal{X}} * C_{\mathcal{X}}^\dagger)^T = C_{\mathcal{X}} * C_{\mathcal{X}}^\dagger$, *and*

(d) $(C_{\mathcal{X}}^{\dagger} * C_{\mathcal{X}})^T = C_{\mathcal{X}}^{\dagger} * C_{\mathcal{X}}$.

4. *If* $\mathbf{X}$ *has full rank, then* $C_{\mathcal{X}}^{\dagger} = C_{\mathcal{X}}^{-1}$.

Proof. We will prove each item as they appear in the theorem.

1. This is true by Lemma 5.2.3, with $\mathcal{A} = \mathcal{X}^T$ and $\mathcal{B} = \mathcal{X}$.

2. We know by [27] that any block circulant matrix can be diagonalized using the discrete Fourier transform, and since $\mathbf{C_X}$ is block circulant, then

$$\begin{aligned}(F_p^* \otimes I_t)\mathbf{C_X}(F_p \otimes I_t) &= \text{diag}(D_1, D_2, \ldots, D_t) \\ \Rightarrow \mathbf{C_X} &= (F_p \otimes I_t)\text{diag}(D_1, D_2, \ldots, D_t)(F_p^* \otimes I_t),\end{aligned}$$

   where $D_j \in \mathbb{C}^{p\times p}$ $1 \leq j \leq t$. Now since $(F_p^* \otimes I_t)$ and $(F_p \otimes I_t)$ are unitary, then

$$\begin{aligned}\mathbf{C_X}^{\dagger} &= (F_p \otimes I_t)[\text{diag}(D_1, D_2, \ldots, D_t)]^{\dagger}(F_p^* \otimes I_t) \\ &= (F_p \otimes I_t)\text{diag}(D_1^{\dagger}, D_2^{\dagger}, \ldots, D_t^{\dagger})(F_p^* \otimes I_t)\end{aligned}$$

3. This is true by Lemma 5.2.4, with $\mathcal{A} = C_{\mathcal{X}}$.

4. If $\mathbf{X}$ has full rank, then $\mathbf{C}$ is symmetric positive definite and thus invertible, so by Lemma 5.2.1, $C_{\mathcal{X}}$ is invertible with its inverse given by $C_{\mathcal{X}}^{\dagger}$.

□

5.3 Estimation

In the previous section, we see the matrix operations and the corresponding 3-tensor operations are connected by the block ciriculant matrix using the slices of 3-tensor along the third mode. In this section, we leverage this connection to provide a least squares estimator of $\mathcal{B}$ for TLM, denoted $\hat{\mathcal{B}}$.

We first define the normal equations for 3-tensors and then show that $\hat{\mathcal{B}}$ solves them. We then motivate the problem from another perspective to derive $\hat{\mathcal{B}}$ by minimizing Frobenius norm of the estimation error.

5.3.1 3-Tensor Normal Equations

Given $X \in \mathbb{R}^{n\times p}$, $\mathbf{y} \in \mathbb{R}^{n\times 1}$, and the matrix equation,

$$X\beta = \mathbf{y},$$

the matrix normal equation is constructed to minimize the sum of square difference between the right and left sides:

$$X^T X\beta = X^T\mathbf{y}.$$

We propose that the 3-tensor version is similarly defined.

Defintion 20. *For $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$ and $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, define the* ***3-tensor normal equation*** *to be*

$$\mathcal{X}^T * \mathcal{X} * \mathcal{B} = \mathcal{X}^T * \mathcal{Y}.$$

Theorem 5.3.1. *The* ***least squares estimator***

$$\hat{\mathcal{B}} = (\mathcal{X}^T * \mathcal{X})^\dagger * \mathcal{X}^T * \mathcal{Y}$$

solves the 3*-tensor normal equation.*

Proof. For the 3-tensor $\mathcal{X}^T$ and $\mathcal{X}$, consider the block circulant matrices constructed using the slices along the third mode, $\mathbf{X^T}$, $\mathbf{X}$. For $\mathcal{B}$ and $\mathcal{Y}$, the equivalent notion is simply their matvec, denoted $\mathbf{b}$ and $\mathbf{y}$. Using the 3-tensor normal equation we have defined, first take the matvec of both sides, yielding

$$\begin{aligned} &\text{matvec}(\mathcal{X}^T * \mathcal{X} * \mathcal{B}) = \text{matvec}(\mathcal{X}^T * \mathcal{Y}) \\ &\qquad \Leftrightarrow \mathbf{X^T X b} = \mathbf{X^T y} \text{ by Lemma 5.2.2 and 5.2.3} \end{aligned}$$

Since this now a matrix normal equation, then the matrix Moore-Penrose Pseudoinverse $\hat{\mathbf{b}} = (\mathbf{X^T X})^\dagger \mathbf{X^T y} \in \mathbb{R}^{nt \times 1}$ solves it as well as minimizes the residual sum of squares [7].

To see how $\hat{\mathcal{B}}$ solves the 3-tensor normal equation, we simply have to form $\hat{\mathcal{B}}$ by folding back $\hat{\mathbf{b}}$. By Theorem 5.2.1, we know that the right folding is $\hat{\mathcal{B}} = \text{fold}_{CS}(\hat{\mathbf{b}}, 2, p \times 1 \times t)$. □

Note that if $\mathbf{X} = \text{circ}(\text{matvec}(\mathcal{X}))$ has full rank, then $(\mathcal{X}^T * \mathcal{X})^{-1} = (\mathcal{X}^T * \mathcal{X})^\dagger$.

5.3.2 Least Frobenius Norm

In this section, we motivate the problem by finding an estimator, to minimize the Frobenius norm of the error $\mathcal{Y} - \mathcal{X} * \mathcal{B}$. We call this estimator

$$\hat{\mathcal{B}}_{min} := \arg\min_{\mathcal{B}} \|\mathcal{Y} - \mathcal{X} * \mathcal{B}\|_F^2.$$

We then show that $\hat{\mathcal{B}}_{min}$, perhaps unsurprisingly, is equivalent to $\hat{\mathcal{B}}$. Rather than using tensor calculus, we derive $\hat{\mathcal{B}}_{min}$ via the Fourier Transform of the original problem.

First we have to define the counterparts of $\mathcal{Y}, \mathcal{X}, \beta$ in Fourier space:

Defintion 21. *Let $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$, $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, and $\mathcal{B} \in \mathbb{R}^{p\times 1\times t}$, then define the Fourier transforms of these 3-tensors as $\tilde{\mathcal{Y}}$, $\tilde{\mathcal{X}}$, and $\tilde{\mathcal{B}}$, where by denoting $\tilde{X}_j := \tilde{\mathcal{X}}[:, :, j]$, we have*

$$\begin{aligned}
\text{matvec}(\tilde{\mathcal{Y}}) &= (F_n \otimes I_t)\text{matvec}(\mathcal{Y}),\\
\text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t) &= (F_n \otimes I_t)\text{circ}(\text{matvec}(\mathcal{X}))(F_p^* \otimes I_t)\\
\text{matvec}(\tilde{\mathcal{B}}) &= (F_p \otimes I_t)\text{matvec}(\mathcal{B}).
\end{aligned}$$

Note that these can be obtained as via:

$$\begin{aligned}
\tilde{\mathcal{Y}}[i, 1, :] &= \text{fft}(\mathcal{Y}[i, 1, :]), 1 \le i \le n,\\
\tilde{\mathcal{X}}[i_1, i_2, :] &= \text{fft}(\mathcal{X}[i_1, i_2, :]), 1 \le i_1 \le n; 1 \le i_2 \le p,\\
\tilde{\mathcal{B}}[i, 1, :] &= \text{fft}(\mathcal{B}[i, 1, :]), 1 \le i \le p.
\end{aligned}$$

We can then continue with the derivation:

$$
\begin{aligned}
\|\mathcal{Y} - \mathcal{X} * \mathcal{B}\|_F^2 &= \|\text{matvec}(\mathcal{Y} - \mathcal{X} * \mathcal{B})\|_F^2 \text{ since the squared Frobenius norm decouples along any mode} \\
&= \|\text{matvec}(\mathcal{Y}) - \text{circ}(\text{matvec}(\mathcal{X}))\text{matvec}(\mathcal{B})\|_F^2 \text{ by definition of } * \\
&= \|(F_n \otimes I_t)[\text{matvec}(\mathcal{Y}) - \text{circ}(\text{matvec}(\mathcal{X}))\text{matvec}(\mathcal{B})]\|_F^2 \\
&\quad \text{since multiplication with unitary matrix preserves the Frobenius norm} \\
&= \|\text{matvec}(\tilde{\mathcal{Y}}) - (F_n \otimes I_t)\underbrace{(F_n^* \otimes I_t)\text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t)(F_p \otimes I_t)}_{=\text{circ}(\text{matvec}(\mathcal{X}))}\text{matvec}(\mathcal{B})\|_F^2 \\
&= \|\text{matvec}(\tilde{\mathcal{Y}}) - \underbrace{(F_n \otimes I_t)(F_n^* \otimes I_t)}_{I_{nt}}\text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t)\text{matvec}(\tilde{\mathcal{B}})\|_F^2 \\
&= \|\text{matvec}(\tilde{\mathcal{Y}}) - \text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t)\text{matvec}(\tilde{\mathcal{B}})\|_F^2 \\
&= \sum_{j=1}^{t} \| \underbrace{\tilde{\mathcal{Y}}[:,:,j]}_{n\times 1} - \underbrace{\tilde{X}_j}_{n\times p}\underbrace{\tilde{\mathcal{B}}[:,:,j]}_{p\times 1} \|_F^2.
\end{aligned}
$$

From this decoupling, we can see that to minimize the sum across the t terms, we can simply minimize each one separately. This is important as it allows us to compute $\hat{\mathcal{B}}_{min}$ very efficiently. Simply transform the problem into Fourier Space, where the block diagonal structure allows us to compute t separate linear regressions (OLS) on the complex values, and then transform the problem back via the inverse FFT. In other words,

$$\hat{\mathcal{B}}_{min}[i_1, i_2, :] = \text{ifft}(\hat{\tilde{\mathcal{B}}}[i_1, i_2, :]), \text{ with}$$

$$\hat{\tilde{\mathcal{B}}}[:,:,j] = (\tilde{X}_j^* \tilde{X}_j)^{-1}\tilde{X}_j^* \tilde{\mathcal{Y}}[:,:,j],$$

where $\tilde{X}_j^*$ is the complex transpose (adjoint) of $\tilde{X}_j$.

The last step is to connect $\tilde{\mathcal{B}}$ from Theorem 5.3.1 with this least Frobenius norm estimator $\tilde{\mathcal{B}}_{min}$.

Theorem 5.3.2. *For $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$ and $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, then the least squares estimator*

$$\hat{\mathcal{B}} = (\mathcal{X}^T * \mathcal{X})^\dagger * \mathcal{X}^T * \mathcal{Y}$$

minimizes the squared Frobenius norm of the error $\mathcal{E} = \mathcal{Y} - \mathcal{X} * \mathcal{B}$.

Proof. It suffices to show that $\hat{\mathcal{B}} = \hat{\mathcal{B}}_{min}$. Now let $\mathbf{X} = \text{circ}(\text{matvec}(\mathcal{X}))$, we then have

$$\begin{aligned}
\mathbf{X} &= (F_n^* \otimes I_t)\text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t)(F_p \otimes I_t) \\
\Rightarrow \mathbf{X}^T = \mathbf{X}^* &= (F_p^* \otimes I_t)\text{diag}(\tilde{X}_1^*, \ldots, \tilde{X}_t^*)(F_n \otimes I_t) \text{ since } \mathbf{X} \text{ is real-valued} \\
\Rightarrow \mathbf{X}^T\mathbf{X} &= (F_p^* \otimes I_t)\text{diag}(\tilde{X}_1^*, \ldots, \tilde{X}_t^*)(F_n \otimes I_t)(F_n^* \otimes I_t)\text{diag}(\tilde{X}_1, \ldots, \tilde{X}_t)(F_p \otimes I_t) \\
&= (F_p^* \otimes I_t)\text{diag}(\tilde{X}_1^*\tilde{X}_1, \ldots, \tilde{X}_t^*\tilde{X}_t)(F_p \otimes I_t) \\
\Rightarrow (\mathbf{X}^T\mathbf{X})^\dagger &= (F_p \otimes I_t)\text{diag}((\tilde{X}_1^*\tilde{X}_1)^\dagger, \ldots, (\tilde{X}_t^*\tilde{X}_t)^\dagger)(F_p^* \otimes I_t) \\
\Rightarrow (\mathbf{X}^T\mathbf{X})^\dagger\mathbf{X}^T &= (F_p^* \otimes I_t)\text{diag}((\tilde{X}_1^*\tilde{X}_1)^\dagger\tilde{X}_1^*, \ldots, (\tilde{X}_t^*\tilde{X}_t)^{-1}\tilde{X}_t^*)(F_n \otimes I_t) \\
\Rightarrow (\mathbf{X}^T\mathbf{X})^\dagger\mathbf{X}^T\text{matvec}(\mathcal{Y}) &= (F_p^* \otimes I_t)\text{diag}((\tilde{X}_1^*\tilde{X}_1)^\dagger\tilde{X}_1^*, \ldots, (\tilde{X}_t^*\tilde{X}_t)^\dagger\tilde{X}_t^*)(F_n \otimes I_t)(F_n^* \otimes I_t)\text{matvec}(\tilde{\mathcal{Y}}) \\
&= (F_p^* \otimes I_t)\text{diag}((\tilde{X}_1^*\tilde{X}_1)^\dagger\tilde{X}_1^*, \ldots, (\tilde{X}_t^*\tilde{X}_t)^\dagger\tilde{X}_t^*)\text{matvec}(\tilde{\mathcal{Y}}) \\
&= \text{matvec}(\tilde{\mathcal{B}}_{min}).
\end{aligned}$$

Now by Theorem 5.2.1, we know that $\text{circ}(\text{matvec}((\mathcal{X}^T * \mathcal{X})^\dagger * \mathcal{X}^T)) = (\mathbf{X}^T\mathbf{X})^\dagger\mathbf{X}^T$, and by the definition of $*$, we then have

$$\begin{aligned}
\text{matvec}(\hat{\mathcal{B}}) &= \text{circ}(\text{matvec}((\mathcal{X}^T * \mathcal{X})^\dagger * \mathcal{X}^T))\text{matvec}\mathcal{Y} \\
&= (\mathbf{X}^T\mathbf{X})^\dagger\mathbf{X}^T\text{matvec}(\mathcal{Y}) \\
&= \text{matvec}(\tilde{\mathcal{B}}_{min}).
\end{aligned}$$

□

We also propose a ridge estimator $\hat{\mathcal{B}}_{ridge}$ that penalizes for the size of the entries in the coefficient matrix $\mathcal{B}$. Specifically, we derive the explicit solution to the estimator that solves the following equation:

$$(\mathcal{X}^T * \mathcal{X} + \lambda \mathcal{I}) * \mathcal{B} = \mathcal{X}^T * \mathcal{Y}. \tag{5.1}$$

We give form to the ridge estimator in the following theorem.

Theorem 5.3.3. $\hat{\mathcal{B}}_{ridge} := (\mathcal{X}^T * \mathcal{X} + \lambda \mathcal{I})^{\dagger} * \mathcal{X}^T * \mathcal{Y}$ *solves Equation 5.1.*

Proof. See proof for Theorem 5.3.1. It suffices to recognize that $\text{circ}(\text{matvec}(\lambda \mathcal{I})) = \lambda I_{nt}$. □

5.4 FFT Estimation Algorithms

In this section, we provide a fast algorithm to compute both $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}_{ridge}$ based on FFT. This algorithm is motivated by the derivation of the least Frobenius norm estimator as well as the algorithms to calculate the t-product and the T-SVD by [27]. Rather than having to explicitly construct, multiply, and invert the block circulant matrices $\mathbf{X} = \text{circ}(\text{matvec}(\mathcal{X}))$, we do the regression step instead in Fourier space. The ridge version of the model is extremely similar.

Note that both are non-iterative algorithms that require

1. $n \times p + n$ FFT operations on vectors of length t,

2. t OLS solves, each with $\tilde{Y} \in \mathbb{C}^{n\times 1}$ and $\tilde{X} \in \mathbb{C}^{n\times p}$,

3. $n \times p$ inverse FFT operations on vectors of length t.

Algorithm 11 Tensor Linear Model (TLM)

input : $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$
for $i_1 = 1, \ldots, n$ **do**
  for $i_2 = 1, \ldots, p$ **do**
    $\tilde{\mathcal{X}}[i_1, i_2, :] = \text{fft}(\mathcal{X}[i_1, i_2, :])$
  end
  $\tilde{\mathcal{Y}}[i_1, 1, :] = \text{fft}(\mathcal{Y}[i_1, 1, :])$
end
for $j = 1, \ldots, t$ **do**
  $\tilde{\mathcal{B}}[:, :, j] = (\tilde{\mathcal{X}}[:, :, j]^* \tilde{\mathcal{X}}[:, :, j])^\dagger \tilde{\mathcal{X}}[:, :, j]^* \tilde{\mathcal{Y}}[:, 1, j]$
end
for $i_1 = 1, \ldots, p$ **do**
  $\hat{\mathcal{B}}[i_1, 1, :] = \text{ifft}(\tilde{\mathcal{B}}[i_1, 1, :])$
end
output : $\hat{\mathcal{B}} \in \mathbb{R}^{p\times 1\times t}$

Algorithm 12 Ridge TLM

input : $\mathcal{X} \in \mathbb{R}^{n\times p\times t}$, $\mathcal{Y} \in \mathbb{R}^{n\times 1\times t}$
, $\lambda \in \mathbb{R}$ **for** $i_1 = 1, \ldots, n$ **do**
  for $i_2 = 1, \ldots, p$ **do**
    $\tilde{\mathcal{X}}[i_1, i_2, :] = \text{fft}(\mathcal{X}[i_1, i_2, :])$
  end
  $\tilde{\mathcal{Y}}[i_1, 1, :] = \text{fft}(\mathcal{Y}[i_1, 1, :])$
end
for $j = 1, \ldots, t$ **do**
  $\tilde{\mathcal{B}}[:, :, j] = (\tilde{\mathcal{X}}[:, :, j]^* \tilde{\mathcal{X}}[:, :, j] + \lambda I_{nt})^\dagger \tilde{\mathcal{X}}[:, :, j]^* \tilde{\mathcal{Y}}[:, 1, j]$
end
for $i_1 = 1, \ldots, p$ **do**
  $\hat{\mathcal{B}}[i_1, 1, :] = \text{ifft}(\tilde{\mathcal{B}}[i_1, 1, :])$
end
output : $\hat{\mathcal{B}} \in \mathbb{R}^{p\times 1\times t}$

5.5 Applications to Functional Data

The vector-variate response of TLM makes functional data analysis a natural application of the model. TLM represents a very different approach than the usual functional data analytic models, since it does not require parameter tuning to adjust the smoothness, the order of the derivatives, or other variable aspects of the model. For a model with more control, the Ridge TLM allows us to adjust the smoothness of our prediction with the λ parameter. We believe that TLM sets up a framework with an application towards functional data analysis, and it is just the basic linear model in this context. Setting this foundation will allow deeper understanding of tensor regression, so that we can begin to model more complex relationships using non-linearity and penalization.

We apply TLM as well as the Ridge TLM to the "Lips" dataset that has a functional response as well as functional covariates, found in the `R` package `fda` [49] and originated with the background paper by Malfait, Ramsay and Froda (2001) [40]. The data consist of 32 records of each of three measurements taken at 501 time points taken to utter the phrase "Say bob again". The goal of the original is to predict the acceleration of the lower lip using the position of the lower lip and the measure of the electromyographical (EMG) activity in the depressor labii inferior (DLI) as the two covariates.

One natural question that may arise in any tensor setup is "which mode corresponds to what?" Recall that for TLM, where $\mathcal{Y} = \mathcal{X} * \mathcal{B} + \mathcal{E}$ and $\mathcal{X} \in \mathbb{R}^{n \times p \times t}$, the first mode of $\mathcal{X}$ corresponds to replication, the second mode corresponds to the number of covariates, while the third mode corresponds to the number of positions (or inherent dimension) of

the vector-variate response. Since we are interested in predicting the lip acceleration curve during the time necessary to say the phrase using the two covariates, this makes $t = 501$ the inherent dimension. The number of predictors is naturally $p = 2$ here, and we have $n = 32$ observations. Consequently, for the "Lips" dataset, we have

$$\mathcal{Y} \in \mathbb{R}^{32\times1\times501},$$
$$\mathcal{X} \in \mathbb{R}^{32\times2\times501}, \text{ and}$$
$$\mathcal{B} \in \mathbb{R}^{2\times1\times501}.$$

TLM will give us the least squares estimate $\hat{\mathcal{B}}$ for $\mathcal{B}$, the 2×501 coefficients that will allow us to construct an entire lip acceleration curve using the two covariates: a lip position curve and a EMG activity curve.

We first plot each of the 32 curves for the response and the two covariates. The red line represents the "mean" curve across all 32 records.

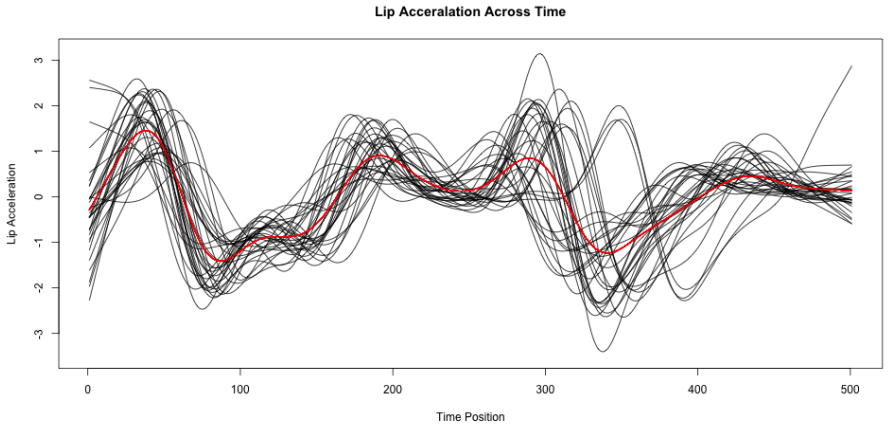


Figure 5.2: Lip Acceleration from the "Lips" Dataset

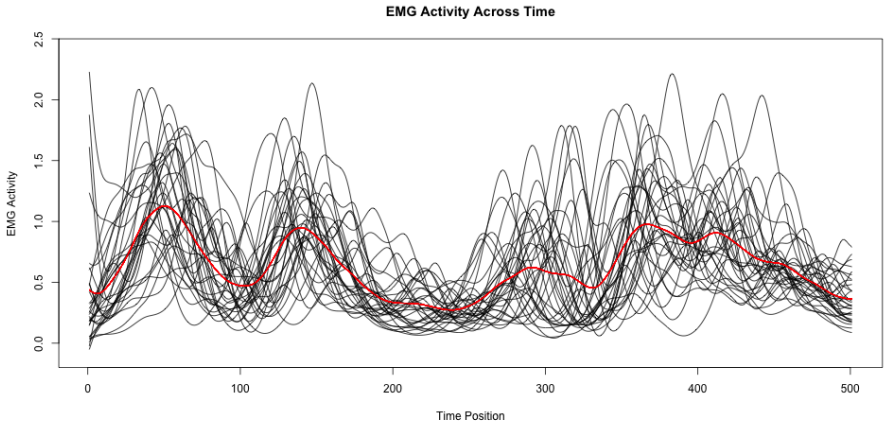


Figure 5.3: EMG Activity from the "Lips" Dataset

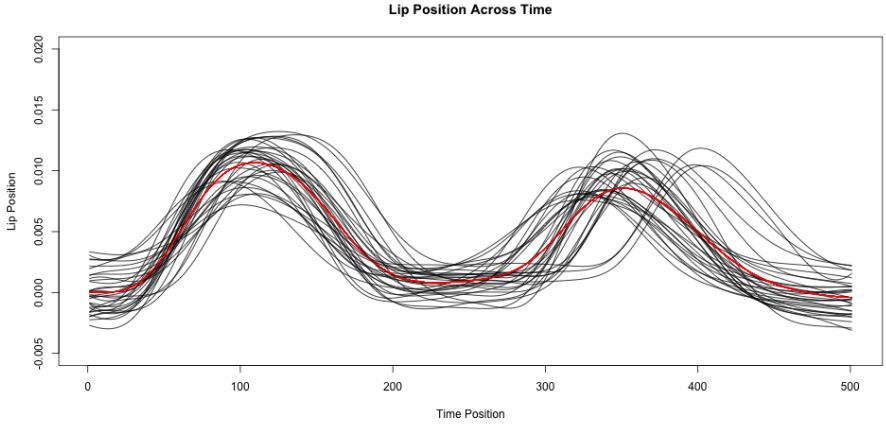


Figure 5.4: Lip Position From the "Lips" Dataset

We then construct $\hat{\mathcal{B}} = (\mathcal{X}^T * \mathcal{X})^{\dagger} * \mathcal{X}^T * \mathcal{Y}$ in the `R` package `rTensor` using the function `tlm`. The estimate is then $\hat{\mathcal{Y}} = \mathcal{X} * \hat{\mathcal{B}} \in \mathbb{R}^{32\times 1\times 501}$. To evaluate our fit, we used the metric $1 - \dfrac{\|\hat{\mathcal{Y}} - \mathcal{Y}\|_F}{\|\mathcal{Y}\|_F}$. Naturally, a smaller Frobenius norm in the estimation error is better, and we want to somehow normalize this norm using the norm of the original response tensor.

For the "Lips" dataset, $\|\hat{\mathcal{Y}} - \mathcal{Y}\|_F = 34.37$ and $1 - \dfrac{\|\hat{\mathcal{Y}} - \mathcal{Y}\|_F}{\|\mathcal{Y}\|_F} = 0.7228$. To get a better sense of the fit, we plotted a few examples with the original curves and the estimates in red.

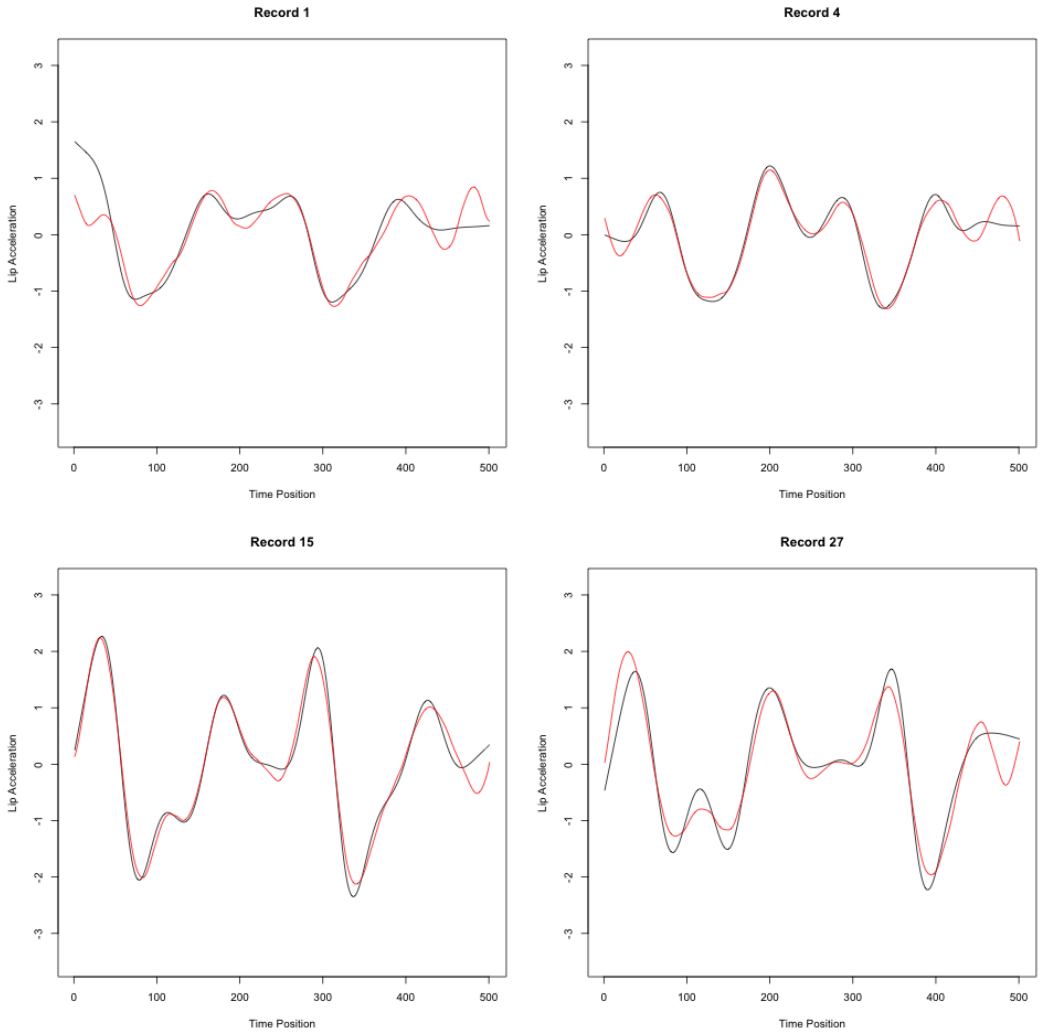


Figure 5.5: Sample Estimated (Red) Curves Compared to the Original (Black) Curves

We also plotted all of the estimated curves side-by-side with the original curves for a

higher-level comparison.

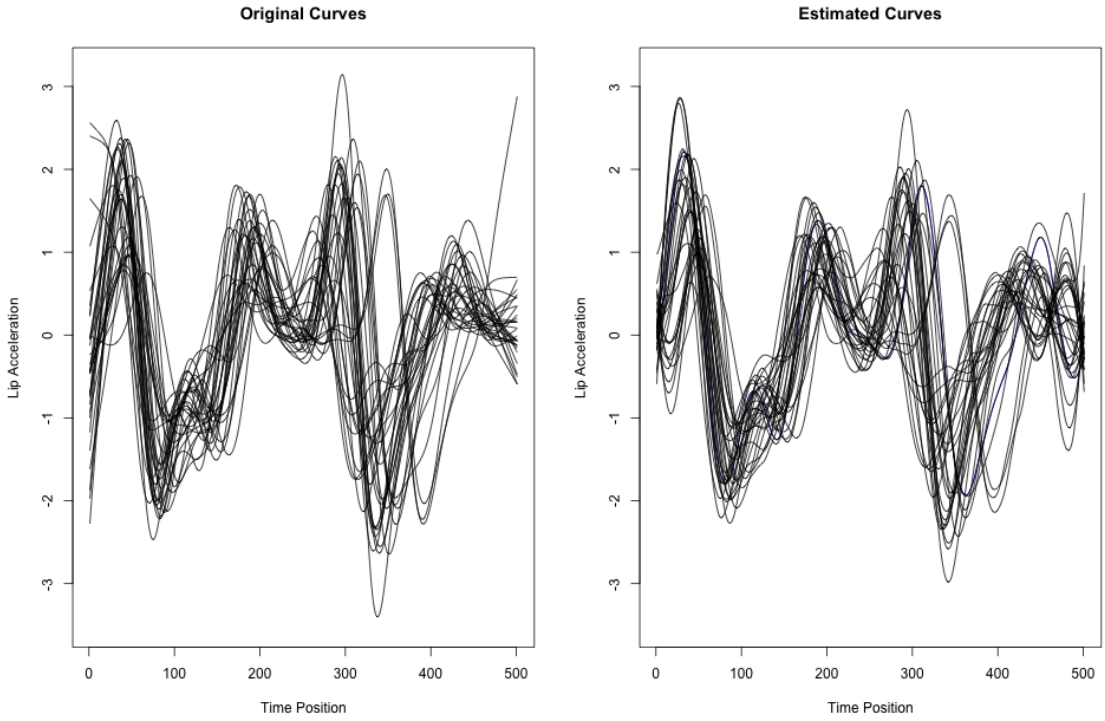


Figure 5.6: All Estimated Curves Compared to All Original Curves

We also compute $\hat{\mathcal{B}}_{ridge} = (\mathcal{X}^T * \mathcal{X} + \lambda * \mathcal{I})^{\dagger} * \mathcal{X}^T * \mathcal{Y}$ using the function `tlm_ridge` across various values of λ, and saw that as the size of λ grew, the estimates were smoother on average, as expected. We demonstrate this with the next plot.

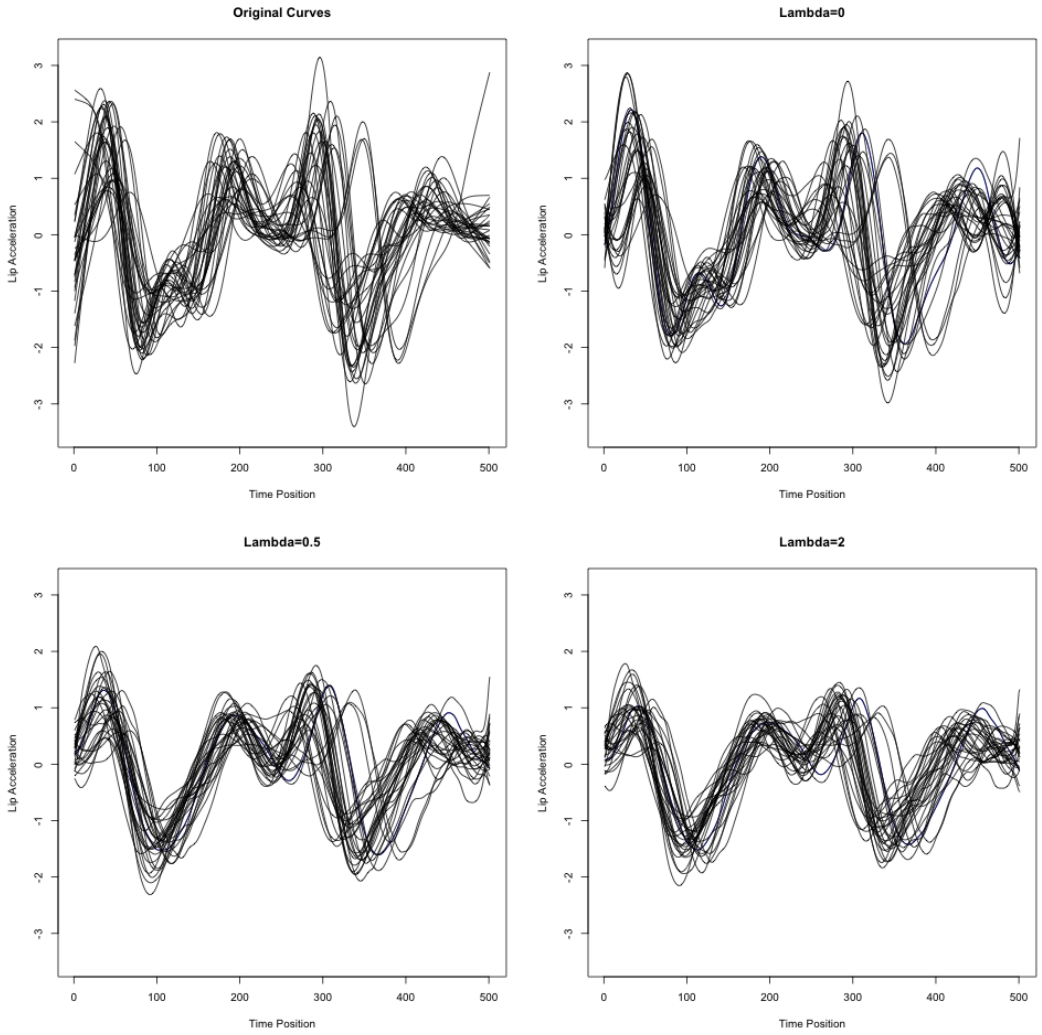


Figure 5.7: Varying Degrees of λ for Ridge TLM

We also used TLM for out-sample predictions with the "Lips" dataset as well. We simply took $\frac{2}{3} * 32 \approx 21$ of the records as training data to predict the remaining 11 based on their covariates. The out-sample error had a Frobenius norm of 30.36, with $1 - \frac{\|\hat{\mathcal{Y}}_{out} - \mathcal{Y}_{out}\|_F}{\|\mathcal{Y}_{out}\|_F} = 0.6173$.

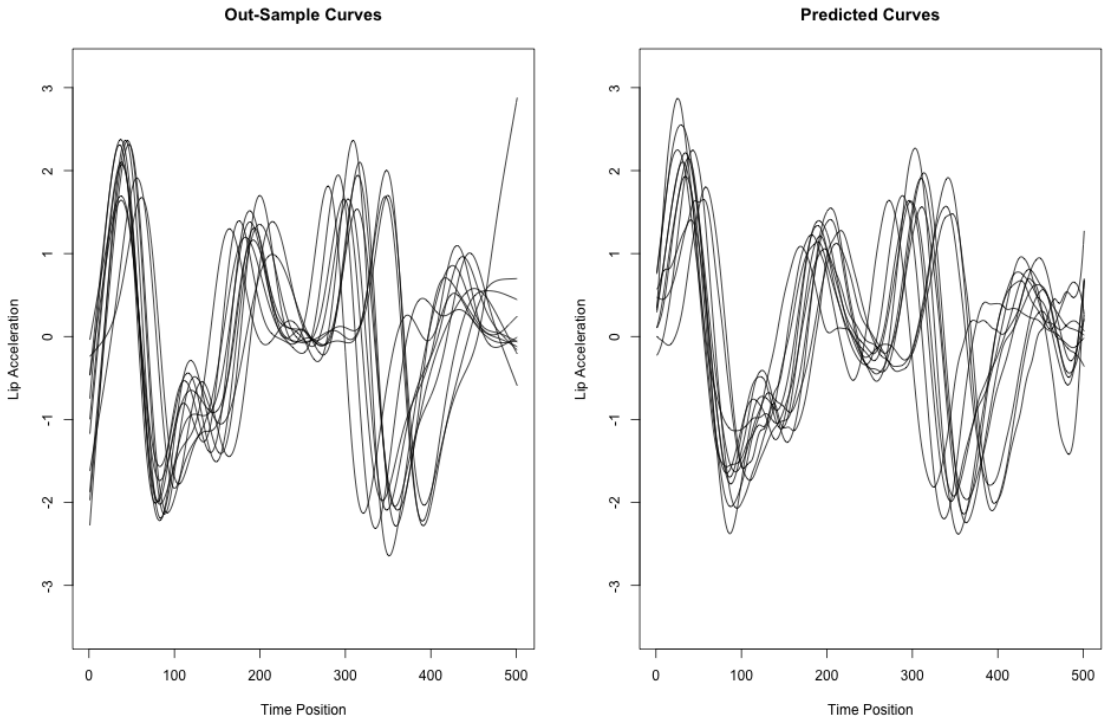


Figure 5.8: All Out-Sample Curves Compared to All Predicted Curves

5.6 Next Steps

There are many important questions that remain unanswered for this novel 3-tensor linear model framework. We provide an outline here and expand on each point.

- **Intercept.** Currently, TLM has no intercept term similar to the constant intercept in ordinary least squares. It would be very interesting to see how the intercept term could be added to design tensor in a reasonable way. One idea is to add

the intercept as a vector of one's to the block circulant design matrix as a result of circ(matvec($\mathcal{X}$)), but this would no longer preserve the block circulant structure, which makes the current FFT-based algorithms invalid.

- **Normality.** It is crucial to understand how the matrix-variate normal fits into the TLM framework. This is necessary to induce an inferential framework from which we can derive standard errors, study the asymptotic behavior of $\hat{\mathcal{B}}$ in all three modes (n, p, and t), and derive statistical tests. In particular, what is the distribution of $\hat{\mathcal{B}}$ if $\mathcal{E}$ is a matrix-variate normal? What if we removed the identification restriction and considered vec($\mathcal{E}$) as a multi-variate normal? The t-product and the corresponding operations suggest that these two assumptions would lead to very different distributional forms of $\hat{\mathcal{B}}$.

  The tensor-variate normality distribution is important in its own right, and may not be the same as the multilinear normal distribution. The relationship between the multivariate normal distribution and the current matrix-variate distribution is well known, but the identification restriction placed on the covariance matrix seems to suggest that there is a more general form of the matrix-variate normal.

  We also need to define the residual sum of squares. With the residuals now being a matrix, there is more than one way to make such a definition. Without a underlying normal theory, it is not clear which one is correct.

- **Multivariate Link Function.** Understanding Normality in the context of TLM will also allow us to extend to the general exponential family, creating the Tensor Generalized Linear Model. To do this, we also need to explore the use of multivariate link functions such as the logistic or probit. In general, these are not as well understood

as the GLM.

- **Asymptotics.** Currently, we are able to empirically validate how $\hat{\mathcal{B}}$ behaves with increasing one of n, p, and t while holding the other two fixed. We saw, perhaps unsurprisingly, in our simulations, that holding p and t fixed and increasing n, gave much better estimates for $\mathcal{B}$. What was surprising is that the same phenomenon occurred when we increased t while holding n and p fixed. The rate at which $\hat{\mathcal{B}}$ approached $\mathcal{B}$ also seemed slower than the previous scenario. Finally, if we fixed n and t, increasing p lead to worse estimates of $\mathcal{B}$.

- **Shrinkage.** With the parameters $\mathcal{B}$ now structured in a matrix form, there seems to be more avenues and possible structures for shrinkage. While we showed that one particular form of ridge still holds true from the matrix version, it is not clear if this is the one way to impose a L2 penalty on the parameters. Furthermore, it was not immediately obvious to us how to impose L1 penalty such as the Lasso. However, with more parameters and much more data than before, it seems that sparse signals are a lot more likely in the tensor data analytic framework. This area is very open and our work suggests that there might be some results that can be easily translated from the matrix case.

CHAPTER 6

TENSOR SOFTWARE

In this chapter, we provide an overview to available software for tensor analysis and decompositions. We also introduce a new R package, `rTensor`, that allows much more direct and flexible modeling of higher-order tensors [36]. While several MATLAB implementations [4, 6] exist, R is still the tool of choice for many statisticians, and we believe that this package will allow for faster prototyping of tensor models for statistics, and help distribute state-of-the-art tensor methods for data analyses.

6.1 Available Tensor Software

Tensor Software is available in multiple platforms. We surveyed and used most of the freely-available ones before deciding to build `rTensor`. In `MATLAB`, Andersson and Bro created "The N-way Toolbox" [4] to fit the CP, Tucker, as well as other multlinear models. This toolbox also handles missing missing values. Bader and Kolda created the "Tensor Toolbox" [6] that provides classes for dense, sparse, and structured tensors. [6] also provides most of the tensor decompositions that we have discussed in this thesis. We have aimed to structure our R package `rTensor` after [6].

In `C++`, there are also several libraries designed for tensor operations. Tensor [50], for example, is optimized for matrix and vector operations implemented on top of ATLAS and BLAS kernels. Boost [18] and Blitz++ [56] also both implement multidimensional

arrays that promise efficiency and large number of dimensions. However, we found these `C++` libraries mainly lacking in the methods we have described as well as tensor objects where the number of modes that can be dynamically altered after compile time. While computational efficiency is certainly important, we needed more flexibility in a tensor software that allowed easy tensor analysis and prototyping of models.

In `R`, there is the base `array` class [48] is a multidimensional array with arbitrary number of dimensions. The package `tensorA` [54] provides Einstein and Riemann summing conventions as well as parallel computations for tensors. It does not support any models and decompositions that have been discussed, however. The package `PTAk` [35] does support CP, general Tucker, as well as 2dPCA, but it does not export a new class that unifies the tensor computational framework.

6.2 Introduction to `rTensor`

`rTensor` is developed with the aim of making tensor operations and decompositions more accessible to R users. It is designed to mirror MATLAB implementations of tensor software, which are fairly extensive, export tensor classes, as well as provide support for prototyping novel tensor methodology. `rTensor` provides a S4 class `Tensor` that allows creation and manipulation of arbitrary-order Tensors. The `Tensor` class extends familiar R multidimensional array subset operations, element-wise operations, permutations, etc. It also implements new functionality for multidimensional arrays, most notably the class of unfolding operations.

`rTensor` also implements all the tensor operations and decompositions we have described in this paper, as well as the regression model Tensor Least Squares. We started with the classic implementations of these algorithms, but plan to introduce other implementations as well in the future. Much of the operations associated with the *t*-product [26] (i.e. tensor transpose, T-SVD, etc.) are implemented for 3-tensors only, since these operations are recursively defined for general K-tensors, and recursion is particularly expensive in R.

Currently, `rTensor` works well for small to medium-sized datasets (e.g. $500 \times 500 \times 100$), but there is still a need to parallelize many of these methods, as well as offer speed improvements by porting some of the internal calculations to lower-level languages such as `C++` or `FORTRAN`, so that tensor datasets of internet-size can be analyzed in reasonable time.

The remainder of this chapter will be devoted to providing details about the functionality of `rTensor`. We will first describe the S4 Class `Tensor` in Section 6.3. We will then show how a tensor object can be constructed and manipulated in Section 6.4. In Section 6.5, we demonstrate how to unfold a K-tensor into a matrix and fold it back from a matrix, as well as show how to perform the two types of tensor multiplication. We then summarize the various tensor decompositions in Section 6.6. Finally, functionalities that are specific to the *t*-product are described Section 6.7, including the Tensor Least Squares model.

6.3 S4 Class

`rTensor` exports the `Tensor` S4 class, which extends the base 'array' class that ships with every version of R. The most accurate way to consider the `Tensor` class is to see it as an API to the default R multidimensional array, allowing the user to easily create, manipulate and model tensors coherent with the set of terminology and algorithms set forth by [26, 55, 28, 10].

R provides three possible Object-Oriented (OO) systems - S3, S4, and Reference Class [47]. Our rationale for choosing S4 over S3 is summarized as follows:

- S4 allows multiple dispatch while S3 can only dispatch on the first argument. For methods that take in at least two tensors as arguments, this can lead to confusion for the user and much less control over what the method can expect.
- The S3 class system is rather informal and there is no internal validation of names and types of components in the object.
- Methods associated with the S4 class system require a formal `signature` for each argument. Along with the possible meta-classes `ANY` and `missing`, S4 provides the right amount of control over what each method of `Tensor` can expect.
- S4 facilitates multiple inherence much better than S3. This is not an obvious advantage for the current implementation, but one future goal is to allow multiple classes to extend a virtual `Tensor` class (i.e. `sparseTensor` or `integerTensor`). Each of these sub-classes will then be optimized with respect to the default operations and decompositions associated with them.

Our rationale for choosing S4 over the Reference Class is more practical. As of writing this thesis, there are many more R packages that uses S4 classes - including the entire BioConductor project [17] - rather than the very new Reference Class. We expect that there may be major upcoming updates to Reference Class system, so S4 was chosen for simplicity and short-term stability. One could argue that by providing the ability to pass an object by reference, which is provided by the Reference Class, is something that could lead to substantial speed improvements as `rTensor` manipulates and rearranges potentially large objects (multidimensional arrays that could have large extents).

The `Tensor` class contains three slots:

Slot Name	Type	Description
`num_modes`	`integer`	The number of modes, or K.
`modes`	`vector`	The vector of modes/sizes/extents/dimensions.
`data`	`vector`, `matrix`, or `array`	The actual data of the tensor.

Figure 6.1: Slots in the `Tensor` S4 Class

We also provide a getter method for each of these slots: `getNumModes`, `getModes`, and `getData`. Since `Tensor` extends `array`, we want to make sure that most of the common methods that are used for `array` are overloaded for `Tensor`. These methods are listed and described in Figure 6.2. In addition, we also provide some extra functionality that we found lacking for purpose of tensor decomposition in the base `array` class. These methods are listed and described in Figure 6.3. Additional functionality and usage of these methods are shown in the later sections.

Method Name	Output Type	Purpose
`dim`	`integer`	Returns the vector of modes. Same as `getModes`.
`head`	`Tensor`	Returns the first n values of the `Tensor` object. Default $n = 6$.
`tail`	`Tensor`	Returns the last n values of the `Tensor` object. Default $n = 6$.
`print`	None	Prints out only the relevant information of the `Tensor` object.
`show`	None	Prints only relevant information when name of the `Tensor` object is called.
`sweep`	`Tensor`	Obtain marginal statistics for the `Tensor` object.
`Ops`	`Tensor`	Overloads element wise operations for two tensors of same modes.
`[`	`Tensor`	Subsets parts of the `Tensor` object to return a sub-tensor.
`tperm`	`Tensor`	`Tensor` version of `aperm`. General reshaping of the `Tensor` object.
`t`	`Tensor`	Tensor transpose for 3-tensors only. See Section 6.7

Figure 6.2: Methods in `array` overwritten by `Tensor`

Method Name	Output Type	Purpose
`fnorm`	`integer`	Returns the Frobenius norm of the `Tensor` object.
`innerProd`	`integer`	Returns the inner product between two `Tensor` objects.
`unfold`	`matrix`	General matrix unfolding of a `Tensor` object. See Section 6.4.
`rs_unfold`	`matrix`	Row space matrix unfolding of a `Tensor` object. See Section 6.4.
`cs_unfold`	`matrix`	Column Space matrix unfolding of a `Tensor` object. See Section 6.4.
`modeSum`	`Tensor`	Sums across a single mode and returns a $(K-1)$-tensor.
`modeMean`	`Tensor`	Takes the mean across a single mode and returns a $(K-1)$-tensor

Figure 6.3: Methods new to `Tensor`

6.4 Creation & Manipulation

In this section, we demonstrate how to create and manipulate a `Tensor` object. The most direct way is to use the `new` function, specifying the three slots directly.

```
> tnsr <- new("Tensor", num_modes = 3L, modes =c(3L, 4L, 5L),
+ data = runif(60L))
```

Since a `Tensor` object is most likely going to be associated with a (multidimensional) dataset, then it is unlikely that users of `rTensor` will be inputting the data directly. Instead, it is most like that users will be looking to turn a dataset that is already a `array`, `matrix`, or `vector` into a `Tensor` object. To this end, we created the function `as.tensor`, which takes in any of those three classes mentioned and returns a `Tensor` object. We expect this to be the main way to create `Tensor` objects.

```
> indices <- c(10, 20, 30, 40)
> arr <- array(rnorm(prod(indices)), dim=indices)
> tnsr <- as.tensor(arr)
```

As an added convenience feature for users who wish to explore the functionality of `rTensor`, we provide the function `rand_tensor()`, which takes in an argument `modes =` (default is `c(3,4,5)`) and returns a `Tensor` with the specified modes, each element coming from i.i.d. standard Gaussian.

We use a particular image dataset as a running example in the remainder of this section, the ORL face database [12]. This dataset contains images of 40 individuals, each taken under one of 10 possible lighting conditions. Each image consists of 92×112 pixels. The data is included in `rTensor` as a 4-tensor with $n_1 = 92$, $n_2 = 112$, $n_3 = 40$, $n_4 = 10$.

```
> require(rTensor)
> faces.tnsr
Numeric Tensor of 4 Modes
Modes:  92 112 40 10
Data:
[1] 0.2000000 0.2000000 0.2000000 0.1764706 0.2039216 0.1960784
```

After the `Tensor` object has been created/loaded, we can subset it just like we would an `array` object. We can also specify whether or not to drop indices where the mode is 1 after subsetting. As an example, we can take the second image of the first individual out of the ORL face database by simply calling `faces.tnsr[, , 1, 2]`. Notice, however, that this returns a 2-tensor as now the first two indices in the original tensor is 1, and `rTensor` drops these by default. To prevent this behavior and keep the image as a 4-tensor, we would need to call `faces.tnsr[, , 1, 2, drop=FALSE]`.

We can also create a tensor of "average" faces across all 10 lighting conditions, one for each of the 40 individuals. To do this, we would simply call

```
> avg. faces <- modeMean(faces.tnsr,m=4,drop=TRUE)
```

, which returns a 3-tensor of size $92 \times 112 \times 40$, each slice along the third dimension being an "average face" for each individual. For instance, if we were to call `avg.faces[, , 17]`, we would get back the "average" face for the seventeenth individual in the dataset.

Another useful function for tensor manipulation is `sweep`. Given a `Tensor` object, a vector of modes, a statistic, and some binary function, `sweep` would return a `Tensor` of the same size resulting from applying the binary function on each sub-tensor along the modes not specified and the statistic. If the statistic is a `vector`, then recycling occurs. If the statistic is anything else, then the dimensions of sub-tensors along the modes not specified should match the dimensions of the statistic. Continuing our example with the ORL faces, suppose we would like to obtain the deviations from the "average faces" for each image. Instead of a nested for-loop, we would call

```
> dev.faces <- sweep(faces.tnsr, m=c(1,2,3), stats=avg.faces,
+ func='-')
```

Here each sub-tensor is every 3-tensor along the fourth mode of `faces.tnsr`, the binary function is element-wise subtraction, the statistic is the 3-tensor "average faces". The `sweep` function then gives us the difference between each sub-tensor and the statistic. Note that for users who are familiar with the `sweep` method for `array` or `matrix`, the functionality is exactly the same for `Tensor`.

Often times, one might wish to convert the `Tensor` object back to the base `R array`, `matrix`, or `vector`. Since the data is actually stored as part of the `Tensor` class, then

one would simply need to call method `getData` to return the actual data in the correct class. This highlights the flexibility of the `Tensor` class, as we could have both inputs and outputs in base `R` objects that other packages know about and can interact with.

6.5 Unfolding & Multiplication

In `rTensor`, we provide the following matrix unfoldings for general K-tensors - the row space unfolding in the mode k (`rs_unfold`), column space unfolding (`cs_unfold`) in the mode k, and the general matrix unfolding (`unfold`), as well as the inverse operations to these unfoldings. Recall from Section 2.3 that these three unfolding encompass the k-mode unfolding and the matvec, since the row space unfolding in the mode k is the same as the k-mode unfolding, and the column space unfolding in the mode 2 is the same as matvec.

To invoke `rs_unfold` on a `Tensor` object, we simply have to specify the mode that will occupy the row space:

```
> tnsr <- rand_tensor(modes=c(2,3,4,5,6))
> rs_unfold(tnsr,m=2)
Numeric Tensor of 2 Modes
Modes:  3 240
> rs_unfold(tnsr,m=4)
Numeric Tensor of 2 Modes
```

```
Modes:  5 144
```

The `rs_fold` method, on the other, requires the modes of the original `Tensor` object, since all it sees is the matrix as the input.

```
> tnsr <- rand_tensor(modes=c(2,3,4,5,6))
> unfolded <- rs_unfold(tnsr,m=4)
> rs_fold(unfolded, m=4, modes=c(2,3,4,5,6))
Numeric Tensor of 5 Modes
Modes:  2 3 4 5 6
> identical(rs_fold(unfolded, m=4, modes=c(2,3,4,5,6)),tnsr)
[1] TRUE
```

The exact same principles apply for `cs_unfold` and the corresponding `cs_fold`, except we would need to specify the mode that will occupy the column space.

Both the `rs_unfold`/`rs_fold` and `cs_unfold`/`cs_fold` are special cases of the more general `unfold`/`fold` methods. The general matrix unfolding maps the a subset of the modes onto the row space and the remaining modes onto the column space. As such, it needs to know both *which modes* get mapped to the row space (`rs=`) and *which modes* get mapped to the column space (`cs=`). The permutation order of the indices within the row space and column space depend on the order given in these two parameters. Consider the following 4-tensor as an example:

```
> tnsr <- rand_tensor(modes=c(3,4,5,6))
```

```
> unfold(tnsr,rs=c(1,2),cs=c(3,4))
Numeric Tensor of 2 Modes
Modes:  12 30
> unfold(tnsr,rs=c(2,3),cs=c(1,4))
Numeric Tensor of 2 Modes
Modes:  20 18
```

In the general `fold`, we would need to specify the full modes of the original `Tensor` object as well as `rs` and `cs`.

```
> tnsr <- rand_tensor(modes=c(3,4,5,6))
> unfolded <- unfold(tnsr,rs=c(2,3),cs=c(1,4))
> folded_back <- fold(unfolded, rs = c(2,3), cs=c(1,4),
+ modes=c(3,4,5,6))
> identical(folded_back,tnsr)
[1] TRUE
```

`rTensor` also provides both the k-mode multiplication discussed in Section 2.4.1 and the t-product discussed in Section 2.4.2. The former is implemented for general K-tensors, while the latter is only implemented for 3-tensors and will be discussed in more detail in Section 6.7.

The function to perform k-mode multiplication is `ttm` (short for "Tensor Times Matrix", the name of a function with similar usage in the MATLAB Tensor Toolbox [6]).

`ttm` takes in a `Tensor` object, a `matrix` object, and the mode for multiplication. It then proceeds to row space unfold the `Tensor` object in the mode specified, perform matrix multiplication with the `matrix` object on the left, and row space fold the resulting matrix back into a `Tensor`. Naturally, the number of columns of the `matrix` object must match the mode specified for the original `Tensor` object.

For many tensor decompositions, there is frequently a need to perform a series of k-mode multiplications using multiple factor matrices. To this end, `ttl` is a function that takes in a single tensor $\mathcal{X}$, a `List` of `matrix` objects $\{M_1, M_2, \ldots, M_n\}$, and a vector of modes $(i_1, i_2, \ldots, i_n)$, then returns the output $\mathcal{X} \times_{i_1} M_1 \times_{i_1} M_1 \times_{i_3} \ldots \times_{i_n} M_n$. The number of columns of each matrix must match the corresponding modes of $\mathcal{X}$.

We include a demonstration of both the `ttm` and `ttl` functions below.

```
> tnsr <- rand_tensor(modes=c(4,6,8,10))
> mat <- matrix(seq(1:12), ncol=6)
> kmode1<- ttm(tnsr=tnsr, mat=mat, m=2)
Numeric Tensor of 4 Modes
Modes:  4 2 8 10
> identical(mat\%*\%rs_unfold(tnsr, m=2)@data,
+ rs_unfold(kmode1, m=2)@data)
[1] TRUE
> mat2 <- matrix(seq(1:24), ncol=8)
> kmode2 <- ttl(tnsr=tnsr, list_mat = list(mat,mat2), ms=c(2,3))
Numeric Tensor of 4 Modes
```

```
Modes:  4 2 3 10
```

6.6 Decompositions

In this section, we discuss the majority of the tensor decompositions implemented in `rTensor`, saving a few that are specific to 3-tensors to Section 6.7. The following table summarizes all of the decompositions and the inputs. These decompositions represent the bulk of the package, and we try to be consistent in the outputs of each decomposition. The output to every function is a standard list containing objects that are relevant to that decomposition.

For decompositions that allow for compression - `cp`, `mpca`, `tucker`, `pvd`, and `t_compress`, the output for each function will all be a list containing:

- `est` - the compressed estimate of the original `Tensor` object.
- `fnorm_resid` - the Frobenius norm of the difference between `est` and the original `Tensor` object.
- `norm_percent` - the percent of the Frobenius norm "recovered" by the compressed estimated. This is calculated as 1 - `fnorm_resid`/`fnorm(tnsr)`.
- `conv` - whether or not the algorithm converged by the maximum iteration (only for the iterative algorithms such as `cp`, `mpca`, and `tucker`).
- `all_resids` - a vector of residuals at each iteration (only for the iterative algorithms).

Function Name	Tensor Size	Other Parameters
`cp`	$n_1 \times n_2 \times \ldots \times n_K$	number of components r, maximum number of iterations, convergence criterion
`mpca`	$n_1 \times n_2 \times \ldots \times n_K$	vector of ranks $\mathbf{r} = (r_1, \ldots, r_{K-1})$, maximum number of iterations, convergence criterion
`tucker`	$n_1 \times n_2 \times \ldots \times n_K$	vector of ranks $\mathbf{r} = (r_1, \ldots, r_K)$, maximum number of iterations, convergence criterion
`pvd`	$n_1 \times n_2 \times n_3$	vector of left ranks $\ell = (\ell_1, \ldots, \ell_{n_3})$, vector of right ranks $\mathbf{h} = (h_1, \ldots, h_{n_3})$, final left rank r_1, final right rank r_2
`hosvd`	$n_1 \times n_2 \times \ldots \times n_K$	None
`t_svd`	$n_1 \times n_2 \times n_3$	None
`t_compress`	$n_1 \times n_2 \times n_3$	truncation index k

Figure 6.4: Tensor Decompositions in `rTensor`

6.6.1 HOSVD

Both the complete and truncated HOSVD as in Algorithm 3 are provided in `rTensor`. We demonstrate these as follows, using a 4-tensor with I.I.D. standard normal entries.

```
> tnsr <- rand_tensor(modes=c(2,4,6,8))
> hosvd1<- hosvd(tnsr)
> hosvd1$U[[1]]%*%t(hosvd1$U[[1]])
              [,1]          [,2]          [,3]          [,4]
[1,] 1.000000e+00  2.012279e-16 2.220446e-16  4.718448e-16
[2,] 2.012279e-16  1.000000e+00 2.775558e-17 -2.775558e-17
[3,] 2.220446e-16  2.775558e-17 1.000000e+00  3.885781e-16
```

```
[4,] 4.718448e-16 -2.775558e-17 3.885781e-16  1.000000e+00
> hosvd2 <- hosvd(tnsr,ranks=c(1,2,3,4))
> 1-hosvd2$fnorm_resid/fnorm(tnsr)
[1] 0.9813828
```

6.6.2 CP

The classical alternating least squares method to compute the CP decomposition of a general K-tensor given by Algorithm 2. The `cp` function returns, in addition to the standard output items described above, `lambdas` and `U_list`. `lambdas` is a vector containing the elements in the super-diagonal core tensor λ, while `U_list` is the list of factor matrices $U_1, \ldots, U_K$. We demonstrate the CP decomposition on one of the subjects (#14) in the ORL face dataset [12].

```
> subject <- faces.tnsr[,,14,]
> greyscale = grey(seq(0,1, length=256))
> cp1 <- cp(subject,num_components=50)
> cp2 <- cp(subject,num_components=10)
> cp1$norm_percent
[1] 0.874415
> cp2$norm_percent
[1] 0.8102908
```

We have chosen the number of components so that the second estimate should be worse than the first, although we see here that the first 10 components already accounted for 81% of the Frobenius norm, with the remaining 40 accounting for an additional 6.3%. We can also examine the compressed estimates and compare them with the original image. We do so for the first image in the set.

```
> par(mfrow=c(1,3), mar=c(0.1,0.1,1,0.1))
> image_slice(subject[,,1], col=greyscale, main="Original")
> image_slice(cp1$est[,,1], col=greyscale, main="50 Components")
> image_slice(cp2$est[,,1], col=greyscale, main="10 Components")
```

Figure 6.5: CP decomposition with 50 components and 10 components on subject 14 in the ORL Face Dataset. Picture 1 shown. *Left:* Original. *Middle:* 50 components. *Right:* 10 components.

6.6.3 PVD

The PVD model (Algorithm 7) is also provided in `rTensor` via the function `pvd`. Recall that the PVD model is not cast as a tensor decomposition, but rather as drawing inference from a series of matrices. As such, we had to choose a convention for the modes. We decided that the input into a PVD model should be a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, with n_3 being the number of images in the series, and each $n_1 \times n_2$ $\mathcal{X}[:, :, j]$ being an image, $1 \leq j \leq n_3$.

We currently have not found a principled method to optimize the parameters required by the PVD model. Recall that for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, there are a possible $2n_3 + 2$ parameters. At each of the n_3 SVD of individual images, ℓ_j and h_j are the truncation indices for the left and right eigenvectors, respectively. These are the `uranks` and `wranks`. At the end, we also have r_1 and r_2, which are the truncation indices for the two final eigenvalue decompositions of the large covariance matrix. These are the parameters `a` and `b` required by the `pvd` function. Empirically, we have found that having $\ell_j > r_1$ or having $h_j > r_2$ resulted in poor fits of the data. To illustrate `pvd`, we return to the ORL Face Dataset, choosing a different subject this time (subject # 8).

```
> subject <- faces.tnsr[,,8,]
> pvd1 <- pvd(subject, uranks=rep(46,10), wranks=rep(56,10),
+ a=46, b=56)
> pvd2 <- pvd(subject, uranks=rep(46,10), wranks=rep(56,10),
+ a=23, b=28)
> pvd1$norm_percent
```

```
[1] 0.9667298
> pvd2$norm_percent
[1] 0.928208
```

Figure 6.6: PVD model with various ranks on subject 8 in the ORL Face Dataset. Picture 4 shown. *Left:* Original. *Middle:* $l_1 = \ldots = l_{n_3} = r_1 = 46, h_1 = \ldots = h_{n_3} = r_2 = 56$. *Right:* $l_1 = \ldots = l_{n_3} = 46, h_1 = \ldots = h_{n_3} = 56, r_1 = 23, r_2 = 28$.

6.6.4 GLRAM

The Tucker decomposition with orthogonal factors, or HOOI, is provided in the function `tucker`, with the algorithm stated in Algorithm 4. MPCA is simply HOOI with one of the modes uncompressed, and that's also provided through the function `mpca`. We use the convention that the last mode is the uncompressed mode. Both of these functions can be applied to general K-tensors. The 3-tensor version of MPCA, GLRAM (Algorithm 5), can also be computed by simply calling `mpca` on a 3-tensor. To illustrate the usage of these functions, we return to the ORL Face Dataset. We first start with GLRAM on a single

subject (subject #21), choosing the ranks to be equal to half in each mode. We then ran GLRAM on the same subject but reducing the ranks by half in each mode for the second run.

```
> glram1 <- mpca(subject, ranks=c(46,56))
> glram1$norm_percent
[1] 0.9565463
> glram2 <- mpca(subject, ranks=c(23,28))
> glram2$norm_percent
[1] 0.9188975
```

Figure 6.7: GLRAM with various ranks on subject 21 in the ORL Face Dataset. Picture 2 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56$. *Right:* $r_1 = 23, r_2 = 28$.

6.6.5 MPCA

We then turn to the entire face database of 40 individuals and run MPCA on the 4-tensor, compressing on the first three modes, using the same r_1 and r_2 as in GLRAM. We can also examine the Frobenius norm recovered using MPCA, and it seems as though having more individuals (and hence having a 4-tensor) did not help in recovery. This is also noticeable from the estimated images.

```
> mpca1 <- mpca(faces.tnsr, ranks=c(46,56,20))
> mpca1$norm_percent
[1] 0.9460537
> mpca2 <- mpca(faces.tnsr, ranks=c(46,56,10))
> mpca2$norm_percent
[1] 0.8356275
```

Figure 6.8: MPCA with various ranks on the entire ORL Face Dataset. Subject 35, picture 8 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56, r_3 = 20$. *Right:* $r_1 = 46, r_2 = 56, r_3 = 10$.

6.6.6 HOOI

Finally, we can apply the more general HOOI (Algorithm 4) on the entire face dataset, compressing on all 4 modes, as demonstrated here. Once again we use two different vectors of ranks, with the each corresponding rank of the second vector being roughly half of the one from the first vector.

```
> tucker1 <- tucker(faces.tnsr,ranks=c(46,56,35,8))
> tucker1$norm_percent
[1] 0.8168891
> tucker2 <- tucker(face.db,ranks=c(23,28,10,3))
> tucker2$norm_percent
[1] 0.7786446
```

Figure 6.9: HOOI with various ranks on the entire ORL Face Dataset. Subject 11, picture 6 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56, r_3 = 35, r_4 = 8$. *Right:* $r_1 = 23, r_2 = 28, r_3 = 10, r_4 = 3$.

These subsections hopefully demonstrated the flexibility and ease of the `Tensor` class, both in the tensor decompositions included in `rTensor` and possibly in other packages as well. It was not our intention to compare the recovery accuracies of the various tensor compression models here.

6.7 *t*-Product Based Operations

In this section, we elaborate on functions in `rTensor` that are based on the novel perspective on tensors put forth by Kilmer et al. [26]. These functions include the the *t*-product, tensor transpose, and T-SVD, all of which are given directly. These functions also include our own regression model, Tensor Least Squares. Currently, such functions are only implemented for 3-tensors, and future plans to improve `rTensor` include generalizing these methods to general K-tensors [42].

Tensor transpose is a method for the `Tensor` class, with the current restriction that the object must have 3 modes. Recall that the transpose of a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is defined to be $\mathcal{X}^T \in \mathbb{R}^{n_2 \times n_1 \times n_3}$, where each slice along the third mode is transposed and the induce along the third mode is reversed from 2 to n_3. We can do this with any `Tensor` object with 3 modes as we would transpose a matrix.

```
> tnsr <- rand_tensor(modes = c(3,4,5))
> tnsr_transpose <- t(tnsr)
> identical(tnsr_transpose[, , 1]@data, t(tnsr[, , 1]@data))
```

```
[1] TRUE
> identical(tnsr_transpose[, , 2]@data, t(tnsr[, , 5]@data))
[1] TRUE
```

The *t*-product is implemented via the function `t_mult`. It takes in two `Tensor` objects, $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, $\mathcal{Y} \in \mathbb{R}^{n_2 \times L \times n_3}$, and returns $\mathcal{X} * \mathcal{Y} \in \mathbb{R}^{n_1 \times L \times n_3}$ (Algorithm 1).

```
> tnsr1 <- rand_tensor(modes = c(3, 4, 5))
> tnsr2 <- rand_tensor(modes = c(4, 6, 5))
> t_mult(tnsr1, tnsr2)
Numeric Tensor of 3 Modes
Modes:  3 6 5
```

T-SVD (Algorithm 8) is provided in `rTensor` as well. It takes in a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and returns a list containing $\mathcal{U} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$, $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, and $\mathcal{V} \in real^{n_2 \times n_2 \times n_3}$. The resulting list can be fed back into `t_svd_reconstruct` to reconstruct the original tensor $\mathcal{X}$. Unfortunately there is round-off error in the FFT functions provided by base R. One future plan is to swap out the base FFT functions for something more stable - such as the `fftw` package [31].

```
> tnsr <- rand_tensor(modes = c(3, 4, 5))
> decomp<- t_svd(tnsr)
> fnorm(tnsr - t_svd_reconstruct(result))
[1] 2.452957e-15
```

CHAPTER 7

CONCLUSION

7.1 Summary

Tensor datasets are becoming more commonplace with the increased awareness and emphasis on "Big Data" by academia and industry, yet statistical understanding of tensor manipulation and models are still far behind those of matrix models.

In the preceding chapters we investigated the current state-of-the-art in tensor decomposition and regression, and began to unify many disparate models under the statistical tensor framework. We connected the k-mode unfolding and the matvec unfolding of K-tensors using the notion of k-vectors as well as the Row Space Unfolding and Column Space Unfolding notation. We further explored the connection between GLRAM and PVD, both statistical models for the analysis of image datasets, and highlighted their theoretical and practical differences. We then noted the emergence of a new type of tensor decomposition based on the linear operation, t-product, as well as the corresponding novel perspective on 3-tensor models, where the different modes are not interchangeable.

We also introduced Tensor Linear Model, which can be seen as a 3-tensor generalization of the ordinary least squares model. We motivated some of the theoretical properties of TLM, as well as provided a Ridge regression version of the model. Through studying the TLM and motivating its construction, we gained additional insight into the t-product and its intimate connection with the block circulant matrix structure. We showed that many

of the theoretical notions that were available in the matrix case are now also available in similar ways under the *t*-product. Examples include the Moore-Penrose Psuedoinverse and the Normal equation. We then showed that certain functional data analysis problems can be a useful application of TLM, and provided an efficient parameter estimation algorithm inspired by the FFT-based algorithms in Kilmer et al [27].

Finally, we introduced and navigated the `R` package `rTensor`, which exports an easy-to-use interface for tensor modeling and manipulation. The current implementation already provides methods to create tensors from other data structures, unfold them into matrices, and fold the matrices back into tensor form. This package also includes all of the decompositions discussed as well as TLM, and we plan to add to its functionality in the future to help distribute tensor methodologies.

7.2 Future Direction

We currently see that there is a divergence in the road: should statistical tensor models continue with the multilinear structure of the *n*-mode product? Or should we start considering the linear models constructed via the *t*-product? We believe the answer is both. Multilinear as well as linear methods both deepen our understanding of tensor data in different ways. The fact that more than one type of multiplication can be well-defined for general *K*-tensors is a testament to how tensor data structures can be manipulated to extract more meaningful relationships that are not available at the matrix level.

One very important missing piece to both multilinear and linear tensor models is a framework to conduct statistical inference, and that requires a rigorous definition of the tensor-variate normal distribution. The multilinear normal distribution we discussed in Chapter 4 needs to be integrated with the multilinear regression models in a convincing manner so that we can extract begin to understand the parameters from an inferential standpoint. Furthermore, preliminary investigations with the matrix-variate normal and the Tensor Linear Model reveal some issues that results from the Kronecker covariance form of the matrix-variate normal. In particular, linear combinations of matrix-variate normals are not guaranteed to be matrix-variate normal. This suggests that there are other ways to construct a tensor-variate normal. This step is necessary to allow non-linear extensions to the Tensor Linear Model in the same spirit generalized linear models [44] generalized linear models.

Currently, one major challenge for multilinear regression models such as the tensor regression and GLAM is to be able to generalize to tensor-variate responses. Similarly for the linear model regression model such as the TLM, it is imperative to extend these results to admit general K-tensors covariates. The recursive nature of the matvec operation on K-tensor suggests that matrix multiplication lies at the core of t-product for any number of modes. Furthermore, with the computational efficiency offered by the Fourier transform technique, it is possible to express higher-order linear models succinctly in Fourier space. Specific future directions for the 3-tensor linear model have already been outlined in Chapter 5.

From a computational perspective, tensor operations such as the k-mode product and

the t-product present new challenges to design fast and scalable computation routines. A possibility of a LAPACK-like routines specifically for tensors has been discussed at length [37], and the recent introduction of the t-product warrant a serious re-consideration of matrix algebra models using the block circulant structure.

BIBLIOGRAPHY

[1] E. Acar, D. Dunlavy, T. Kolda, and M. Morup. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41 – 56, 2011.

[2] E. Acar, T. Kolda, and D. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations, 2011.

[3] A. Anandkumar, R. Ge, D. Hsu, S. Kakade, and M. Telgarsky. Tensor decomposition for learning latent variable models.

[4] C.A. Andersson and R. Bro. The n-way toolbox for {MATLAB}. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1 – 4, 2000.

[5] C. J. Appellof and E. R. Davidson. Strategies for analyzing data form video fluorometric monitoring of liquid chromatographic effluents. *Analytical Chemistry*, 53:2053–2056, 1981.

[6] B. Bader and T. Kolda. Matlab tensor classes for fast algorithm prototyping. Technical report, Sandia National Laboratories, October 2004.

[7] R.B. Bapat. *Linear Algebra and Linear Models*. Springer, 2nd edition, 2000.

[8] C. L. Bell. Generalized inverses of circulant and generalized circulant matrices. *Linear Algebra and its Applications*, 1(39):133–142, 1981.

[9] E. Oran Brigham. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[10] Rasmus Bro. Parafac. tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149 – 171, 1997.

[11] Rasmus Bro and Henk A. L. Kiers. A new efficient method for determining the number of components in parafac models. *Journal of Chemometrics*, 17(5):274–286, 2003.

[12] AT&T Laboratories Cambridge. *AT&T "The Database of Faces" (formerly "The ORL Database of Faces"*, 1994.

[13] J.Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970.

[14] C. Crainiceanu, B. Caffo, S. Luo, V. Zipunnikov, and N. Punjabi. Population value decomposition: a framework for the analysis of image populations. *Journal of the American Statistical Association*, 106(495):775–790, 2013.

[15] I. Currie, M. Durban, and P. Eilers. Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society: Series B*, 68(2):259–280, 2006.

[16] P. Dutilleul. The mle algorithm for the matrix normal distribution. *Journal of Statistical Computation and Simulation*, 64:105–123, 1999.

[17] R. C. Gentleman et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5, 2004.

[18] R Garcia, J. Siek, and A. Lumsdaine. *The Boost Multidimensional Array Library*, 2001.

[19] Gene H. Golub and Charles van Loan. *Matrix Computations*. John Hopkins University Press, 4th edition, 2012.

[20] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques, 2013.

[21] A. Gupta and D. Nagar. *Matrix Variate Distributions*. Chapman and Hall/CRC, 1 edition, 1999.

[22] N. Hao, M. Kilmer, K. Braman, and R. Hoover. Facial recognition using tensor-tensor decomposition. *SIAM Journal on Imaging Sciences*, 6(1):437–463, 2013.

[23] A. Harshman. Foundations of the parafac procedure: Models and conditions for an explanatory multi-modal factor analysis. *UCLA working papers in phonetics*, 16:1–84, 1970.

[24] P. Hoff. Separable covariance arrays via the tucker product, with applications to multivariate relational data. *Bayesian Analysis*, 6(2):179–196, 2011.

[25] U. Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 316–324, New York, NY, USA, 2012. ACM.

[26] M. Kilmer, K. Braman, N. Hao, and R. Hoover. Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging. *SIAM Journal on Matrix Analysis and Applications*, 34(1):148–172, 2013.

[27] M. Kilmer and C. Martin. Facial recognition using tensor-tensor decomposition. *SIAM Linear Algebra and its Applications*, 435(3):641–658, 2011.

[28] T. Kolda. Multilinear operators for higher-order decompositions. Technical report, Sandia National Laboratories, April 2006.

[29] Tamara G. Kolda. A counterexample to the possibility of an extension of the eckart-young low-rank approximation theorem for the orthogonal rank tensor decomposition. *SIAM J. Matrix Analysis Applications*, 24(3):762–767, 2003.

[30] T.G. Kolda and B.W. Bader. Tensor decomposition and applications. *Society for Industrial an Applied Mathematics*, 51(3):455–500, 2009.

[31] S. Krey, U. Ligges, and O. Mersmann. *Fast FFT and DCT based on FFTW*, 2011. R package version 1.0-3.

[32] Pieter M. Kroonenberg. *Applied Multiway Data Analysis*. Wiley Publishing, 1st edition, 2012.

[33] L. Lathauwer, B. De Moor, and J. Vanderwalle. A multilinear singular value decomposition. *Journal of Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[34] L. Lathauwer, B. De Moor, and J. Vanderwalle. On the best rank-1 and rank(r_1, r_2,. . .,r_n) approximation of higher-order tensors. *Journal of Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[35] Didier Leibovici. *PTAk: Principal Tensor Analysis on k modes*, 2013. R package version 1.2-6.

[36] J. Li, M. Wells, and J. Bien. *rTensor: Tools for Tensor Analysis and Decomposition*, 2014. R package version 1.1.

[37] C. Van Loan. Workshop report. In *Future directions in tensor-based computation and modeling*, May 2009.

[38] H. Lu, K. Plataniotis, and A. Venetsanopoulos. Mpca: Multilinear principal component analysis of tensor objects. *IEEE Trans. Neural Networks*, 19(1):18–39, 2008.

[39] Haiping Lu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, 44(7):1540 – 1551, 2011.

[40] Nicole Malfait and James O. Ramsay. The historical functional linear model. *Canadian Journal of Statistics*, 31(2):115–128, 2003.

[41] K. Mardia, J. Kent, and J. Bibby. *Multivariate Analysis*. Academic Press, 1 edition, 1976.

[42] C. Martin, R. Shafer, and B. LaRue. An order-p tensor factorization with applications in imaging. *SIAM Journal on Scientific Computing*, 35(1):A474–A490, 2013.

[43] T. D. Mazancourt and D. Gerlic. The inverse of a block-circulant matrix. *IEEE Transactions on Antennas and Propagation*, 31(5), 1983.

[44] P. McCullagh and J Nelder. *Generalized Linear Models*. Chapman and Hall/CRC, 2nd edition, 1989.

[45] M. Morup. Applications of tensor (multiway array) factorizations and decompositions in data mining. *WIREs Data Mining Knowledge Discovery*, 1:24–40, 2011.

[46] M. Ohlson, M. Rauf Ahmad, and D. von Rosen. The multilinear normal distribution: Introduction and some basic properties. *Journal of Multivariate Analysis*, 113:37–47, 2013.

[47] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[48] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[49] J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker. *fda: Functional Data Analysis*, 2013. R package version 2.4.0.

[50] J. Ripoll. *Tensor C++ Library*, 1999.

[51] F. Roemer and M. Haardt. A closed-form solution for parallel factor (parafac) analysis. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 2365–2368, 2008.

[52] B. Sheehan and Y. Saad. Higher order orthogonal iteration of tensors (hooi) and its relation to pca and glram. In *SDM*, 2007.

[53] LedyardR Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[54] K. van den Boogaart. *tensorA: Advanced tensors arithmetic with named indices*, 2010. R package version 0.36.

[55] M. Vasilescu. *A Multilinear (Tensor) Algebraic Framework for Computer Graphics, Computer Vision, and Machine Learning*. PhD thesis, Department of Computer Science, University of Toronto, 2009.

[56] T. Veldhuizen, J. Cummings, P. Guio, A. Stokes, and S. Shende. *The Blitz++ Library*, 2011.

[57] R. Vescovo. Inversion of block-circulant matrices and circular array approach. *IEEE Transactions on Antennas and Propagation*, 45(10), 1997.

[58] J. Yang, D. Zhang, A. Frangi, and J. Yang. Two-dimensional pca: A new approach to appearance-based face representation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):131–137, 2004.

[59] J. Ye. Generalized low rank approximations of matrices. *Machine Learning*, 2005.

[60] Kenan Y. Ylmaz, Ali Taylan Cemgil, and Umut Simsekli. Generalised coupled tensor factorisation. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2151–2159. NIPS, 2011.

[61] D. Zhang and Z. Zhou. Two-directional two-dimensional pca for efficient face representation and recognition. *Neurocomputing*, 69:224–231, 2005.

[62] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer. Novel factorization strategies for higher order tensors: Implications for compression and recovery of multi-linear data, 2013.

[63] Qibin Zhao, Cesar F. Caiafa, Danilo P. Mandic, Liqing Zhang, Tonio Ball, Andreas Schulze-bonhage, and Andrzej S. Cichocki. Multilinear subspace regression: An orthogonal tensor decomposition approach. In *Advances in Neural Information Processing Systems 24*, pages 1269–1277. 2011.

[64] H. Zhou and L. Li. Regularized matrix regression. *Journal of Royal Statistical Society, Series B*, 74, 2014.

[65] H. Zhou, L. Li, and H. Zhu. Tensor regression with applications in neuroimaging data analysis, 2012.